Introduction to Gaussian Processes

Prof. Nicholas Zabaras
Center for Informatics and Computational Science
https://cics.nd.edu/
University of Notre Dame
Notre Dame, Indiana, USA

Email: nzabaras@gmail.com
URL: https://www.zabaras.com/

April 15, 2019

Contents

- Matlab Software, What is a Gaussian Process
- Linear Regression Revisited, Gaussian Process Mean and Covariance, SE and Exponential Kernels, Taking Samples from a GP
- Gaussian Process Regression, Marginal Likelihood, Predictive Distribution, Posterior GP, Restrictions on the Kernel Matrix, Recovering Standard Regression for Kernels defined in terms of Basis Functions,
- Learning the hyper-parameters, Automatic relevance detection
- Applications of GPs, Difficulties using GPs, More on Covariance Functions
- Gaussian Process Classification, Laplace Approximation, Connections to Bayesian Neural Networks

Following closely: Bishop CM, *Pattern Recognition and Machine Learning*, Springer, 2006 (Chapter 6)

Matlab Software

There is a very good implementation of GP's for Matlab, which we will be using to generate examples.

It is developed by Rasmussen and Williams (and others) and can be obtained from here:

Documentation for GPML Matlab Code

To run scripts you need to:

1. Download GPML.
2. Unzip it.
3. Open Matlab and CD in GPML folder.
4. Run the matlab command:

```
>> startup
```

Do not forget to do this because nothing will work otherwise…

What is a Gaussian Process?

The informal answer:

A Gaussian Process (GP) is a generalization of a multivariate Gaussian distribution to infinitely many variables.

The formal answer:

A GP is a collection of random variables, any finite number of which have (consistent) Gaussian distributions.

We will explain what this means.

Gaussian Processes

- The idea is similar to linear regression with a fixed set of basis functions:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

- However, here we forget about the parametric model $y(\boldsymbol{x}, \boldsymbol{w})$
- Instead we take an infinite number of basis functions given by a probability distribution over functions.
- This is not difficult to model since we only have to consider the values of the functions at training/test data points.
- Gaussian process models include:
  - Kriging
  - ARMA (autoregressive moving average)
  - Kalman filters
  - Radial basis function networks , etc.

Gaussian Processes

- Models equivalent to Gaussian processes have been widely studied in many different fields.
- In the geostatistics, Gaussian process regression is known as *kriging*.
- Similarly, ARMA (autoregressive moving average) models
- Kalman filters and radial basis function networks can all be viewed as forms of Gaussian process models.

- Cressie, N. (1993). *Statistics for Spatial Data*.Wiley.
- MacKay, D. J. C. (1998). Introduction to Gaussian processes. In C. M. Bishop (Ed.), *Neural Networks and Machine Learning*, pp. 133–166. Springer.
- Williams, C. K. I. (1999). Prediction with Gaussian processes: from linear regression to linear prediction and beyond. In M. I. Jordan (Ed.), *Learning in Graphical Models*, pp. 599–621. MIT Press.MacKay (2003)
- Rasmussen, C. E. (1996). *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. Ph. D. thesis, University of Toronto.
- Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press.Gibbs, 1997;

Linear Regression Revisited

- Consider a model defined by linear combination of M fixed basis functions:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

- A Gaussian prior distribution over the weight vector $\boldsymbol{w}$

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid 0, \alpha^{-1}\boldsymbol{I})$$

governed by the hyperparameter (precision) a induces then a probability distribution over functions $y(\boldsymbol{x})$.

Linear Regression Revisited

- Evaluating $y(\boldsymbol{x})$ for a set of training data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, we have a joint distribution of $y(\boldsymbol{x_1}), \ldots, y(\boldsymbol{x_N})$

$$\boldsymbol{y} = \boldsymbol{\Phi w}, \text{ with elements } y_n = y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)$$

where $\boldsymbol{\Phi}$ is the design matrix with elements $\Phi_{nk} = \boldsymbol{\phi}_k(\boldsymbol{x}_n)$

- $p(\boldsymbol{y})$ is Gaussian, and its mean and covariance can be shown to be

$$\mathbb{E}[\boldsymbol{y}] = \boldsymbol{\Phi}\mathbb{E}[\boldsymbol{w}] = 0$$

$$\operatorname{cov}[\boldsymbol{y}] = \mathbb{E}\left[\boldsymbol{y}\boldsymbol{y}^T\right] = \boldsymbol{\Phi}\mathbb{E}\left[\boldsymbol{w}\boldsymbol{w}^T\right]\boldsymbol{\Phi}^T \quad \Rightarrow \operatorname{cov}[\boldsymbol{y}] = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{K}$$

where $\boldsymbol{K}$ is the Gram matrix with elements defined in terms of the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ as

$$K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\boldsymbol{\phi}(x_n)^T \boldsymbol{\phi}(x_m)$$

- Up to now we only took data points + prior but no target values.

Gaussian Processes: Mean and Covariance

- The model we have seen is one example of a Gaussian process
- Gaussian process is defined as a probability distribution over functions $y(\boldsymbol{x})$ such that the set of values of y($\mathbf{x}$) evaluated at an arbitrary set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ jointly have a Gaussian distribution (in 2D, we have a Gaussian Random Field).
- Key point: **the joint distribution is defined completely by second-order statistics (mean, covariance)**
- Since usually the mean is taken to be zero, we only need the covariance, i.e., the kernel-function:

$$\mathbb{E}\left[y(\mathbf{x}_n) y(\mathbf{x}_m) \right] = k(\mathbf{x}_n, \mathbf{x}_m)$$

- **Instead of choosing (a limited number of) basis functions, we can directly choose a kernel function (which may result in an infinite number of basis functions)**

Gaussian Processes: Mean and Covariance

A multivariate Gaussian is completely specified by its **mean** and its **variance**:

$$\boldsymbol{y} = (y_1, \dots, y_n) \sim \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma}).$$

For a GP:

1. The discrete indices of $\boldsymbol{y}$ become **continuous** $y_{\boldsymbol{x}}$, where $\boldsymbol{x} \in \mathcal{X}$ (some space).
2. You need a **mean function** $m(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$.
3. So, you would say that $f(\cdot)$ is a GP, and we will write:

$$f(\cdot) \sim GP\big(m(\cdot), k(\cdot,\cdot)\big),$$

if for any finite number of $\boldsymbol{x}$'s, $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, the probability distribution of the vector $\big(f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_n)\big)$ is distributed according to:

$$\big(f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_n)\big) \sim \mathcal{N}\left(\begin{pmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_n) \end{pmatrix}, \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}\right).$$

Restrictions on the Covariance

- The only restriction is that the resulting covariance matrix:

$$\begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix},$$

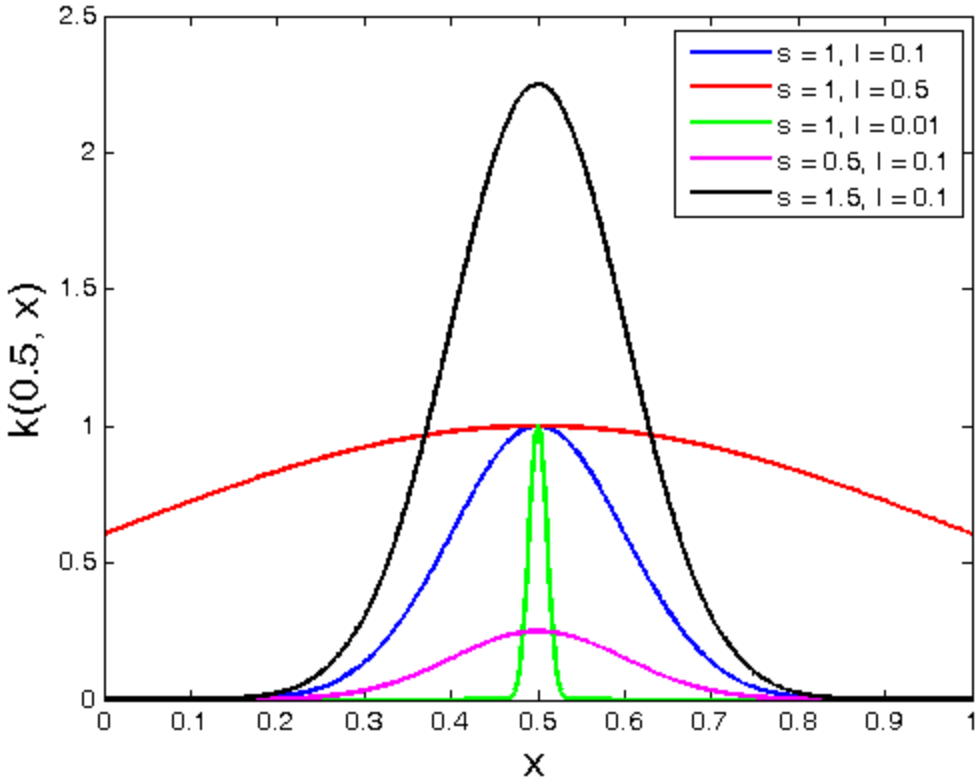
  is *positive definite* for any $\boldsymbol{x}$'s so that the finite dimensional Gaussians are well-defined.

- If that holds, we will say that the function $k(\cdot,\cdot)$ is positive definite.

- A very common choice is the Squared Exponential (SE):

$$k(\boldsymbol{x}, \boldsymbol{x}') = s^2 \exp\left\{ -\frac{1}{2} \sum_{i=1}^{d} \frac{(x_i - x_i')^2}{l_i^2} \right\}.$$

SE Covariance 1D



$$k(\boldsymbol{x}, \boldsymbol{x}') = s^2 \exp\left\{-\frac{(x - x')^2}{2l^2}\right\}.$$

Matlab code: covariance_demo_1d.m

Purpose: Familiarize yourself with GPML covariances and their hyper-parameters.

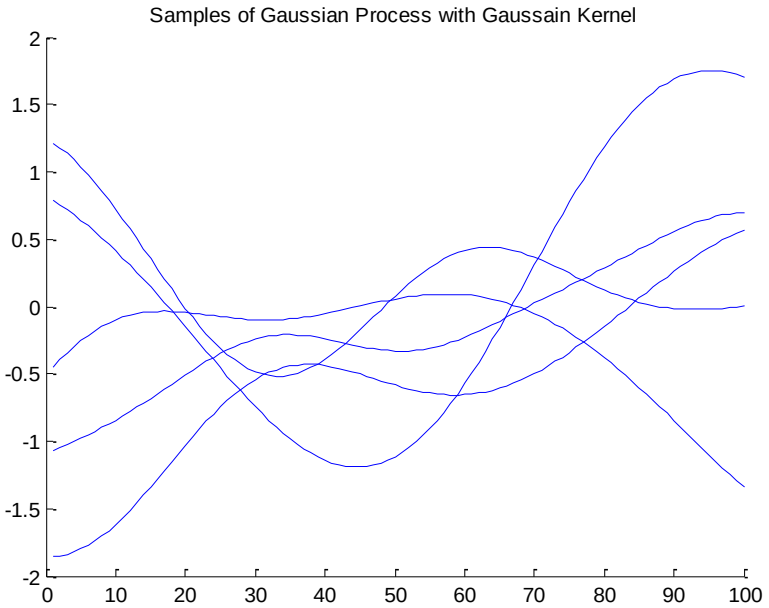
SE and Exponential Kernels

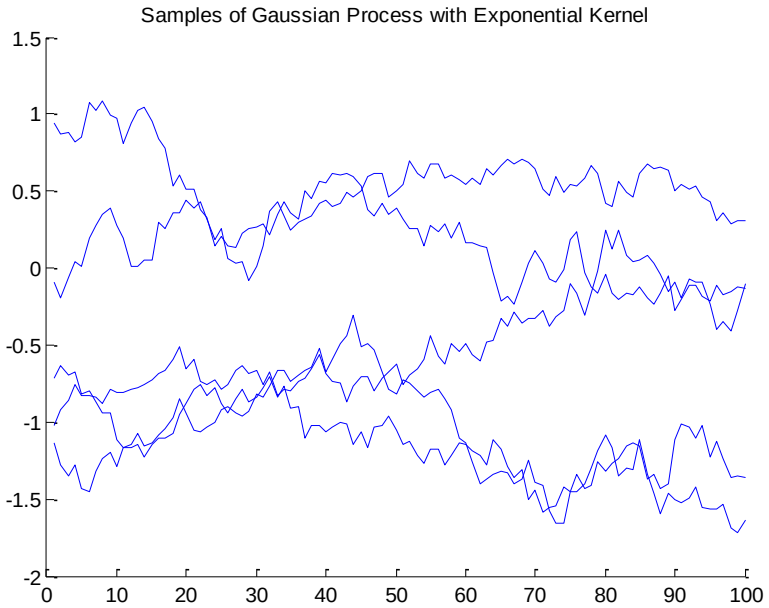
- We show examples of functions drawn from Gaussian processes for a Gaussian kernel and an exponential kernel.

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right)$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\theta\,|\,x - x'\,|\right)$$

(Ornstein-Uhlenbeck process)

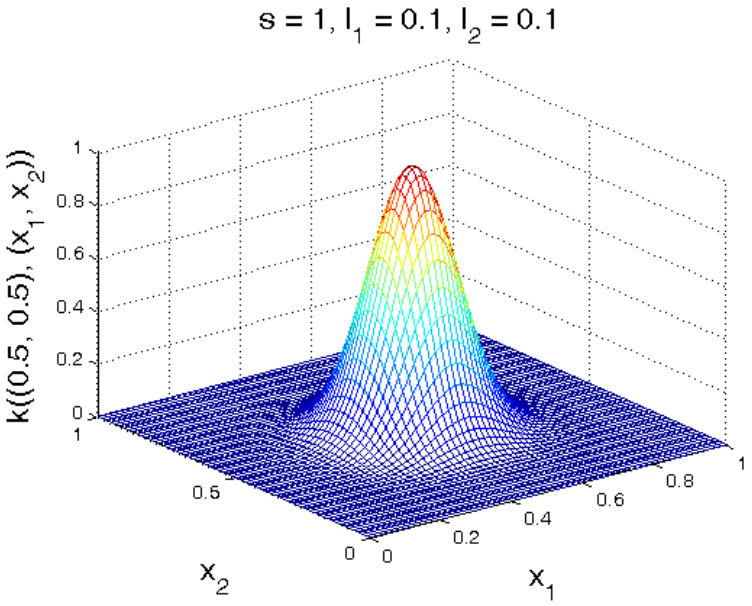




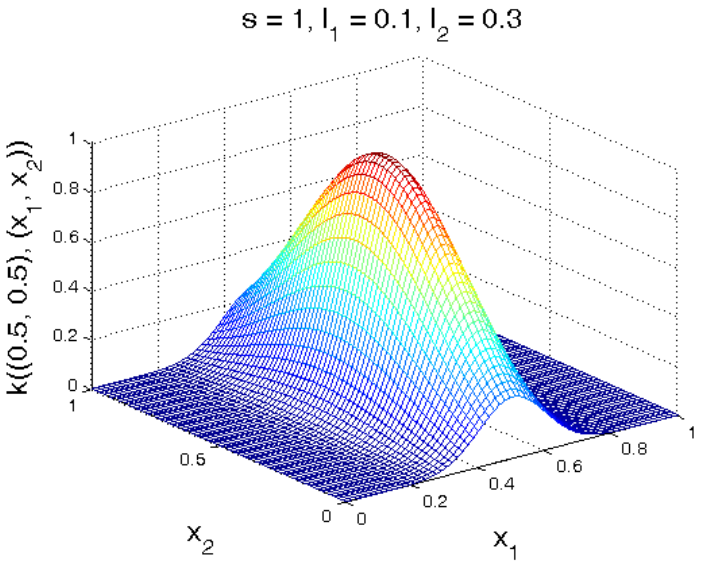
MatLab code

- Uhlenbeck, G. E. and L. S. Ornstein (1930). On the theory of Brownian motion. *Phys. Rev.* **36**, 823– 841.

SE Covariance 2D



Isotropic



Non-isotropic

$$k(\boldsymbol{x}, \boldsymbol{x}') = s^2 \exp\left\{-\frac{1}{2}\sum_{i=1}^{d}\frac{(x_i - x_i')^2}{l_i^2}\right\}.$$

Matlab code: covariance_demo_2d.m

Purpose: Familiarize yourself with GPML covariances and their hyper-parameters.

Taking Samples from a GP

- You can only do it for a finite number of $\boldsymbol{x}$'s, i.e. you have to take samples from a multivariate normal.

- You have to compute the covariance matrix:

$$\boldsymbol{A} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}.$$

- Compute the Cholesky decomposition of $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T.$$

- Compute the mean $\boldsymbol{m} = \big(m(\boldsymbol{x}_1), \dots, m(\boldsymbol{x}_n)\big)$.

- Sample n standard normals:

$$\boldsymbol{z} \sim \mathcal{N}_n(0, I_n).$$

- Compute:

$$\boldsymbol{y} = \boldsymbol{m} + \boldsymbol{L}\boldsymbol{z}.$$

Sample from GP (1D)

Matlab code: take_gp_samples_1d.m

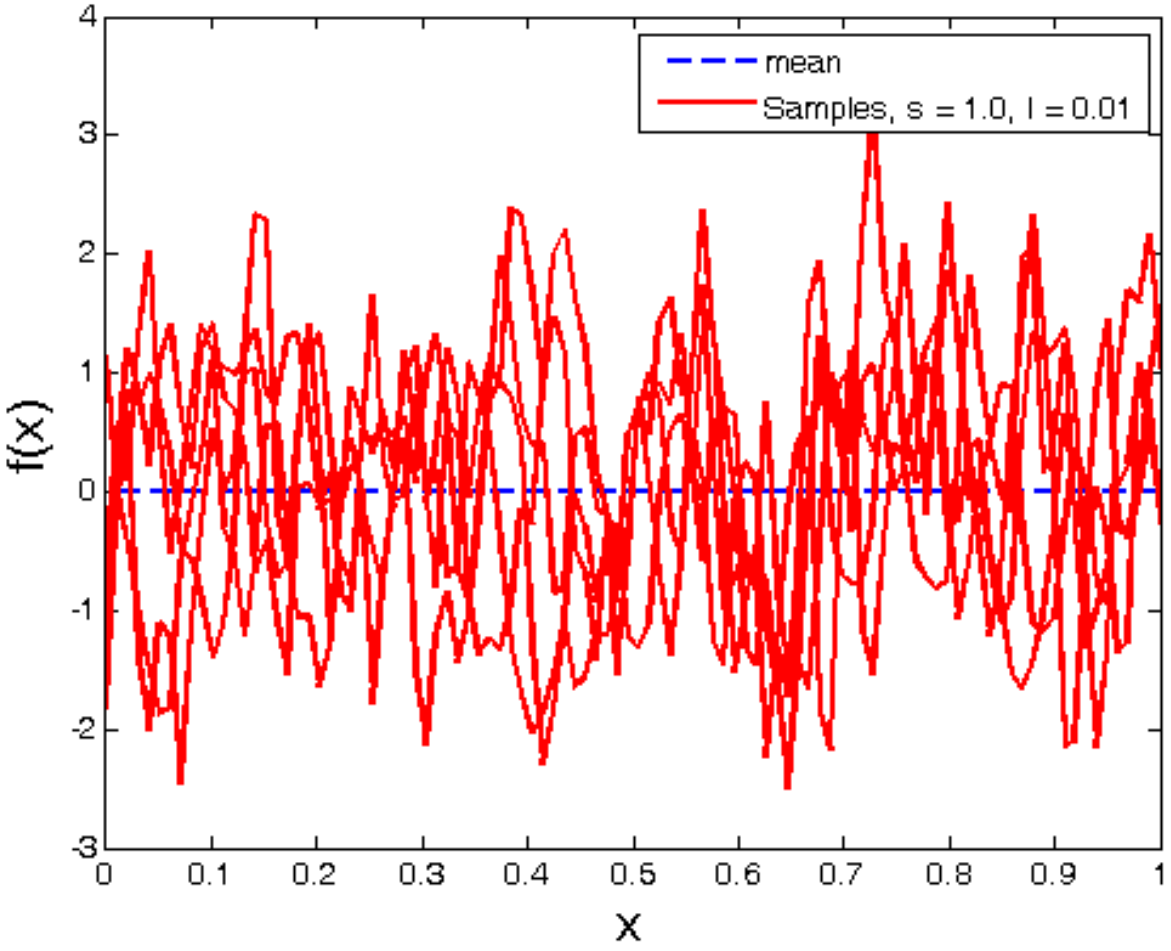
Purpose: Familiarize yourself with GP sampling (essentially sampling from a multivariate Gaussian).

Important Technical Note:

When the length scale of the covariance is large, you might have to add something positive (but small) at the diagonal of the covariance matrix in order to ensure that it is positive definite.

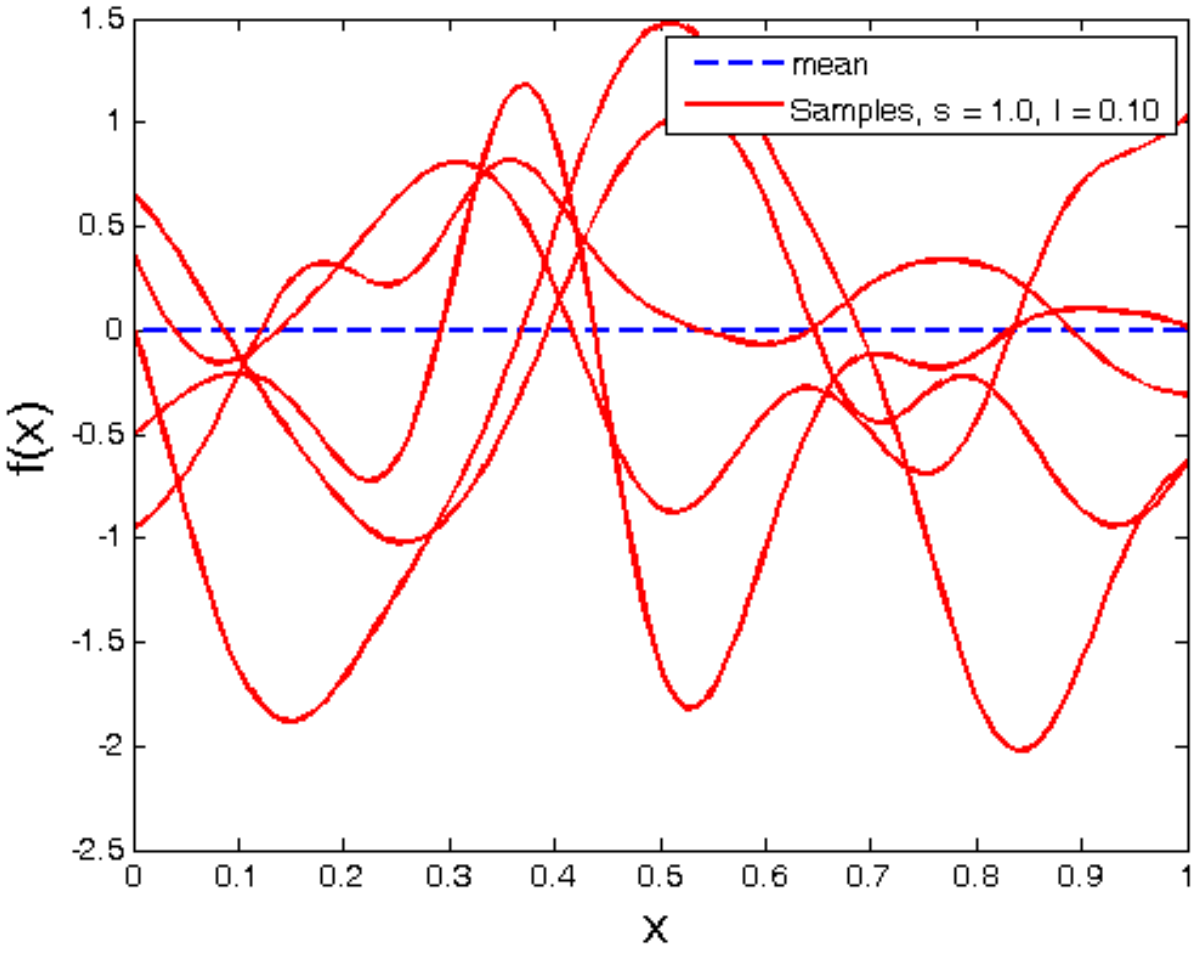
There are ways around this such as making use of the eigen-decomposition of the covariance matrix or the incomplete Cholesky decomposition.

Sample from GP (1D)



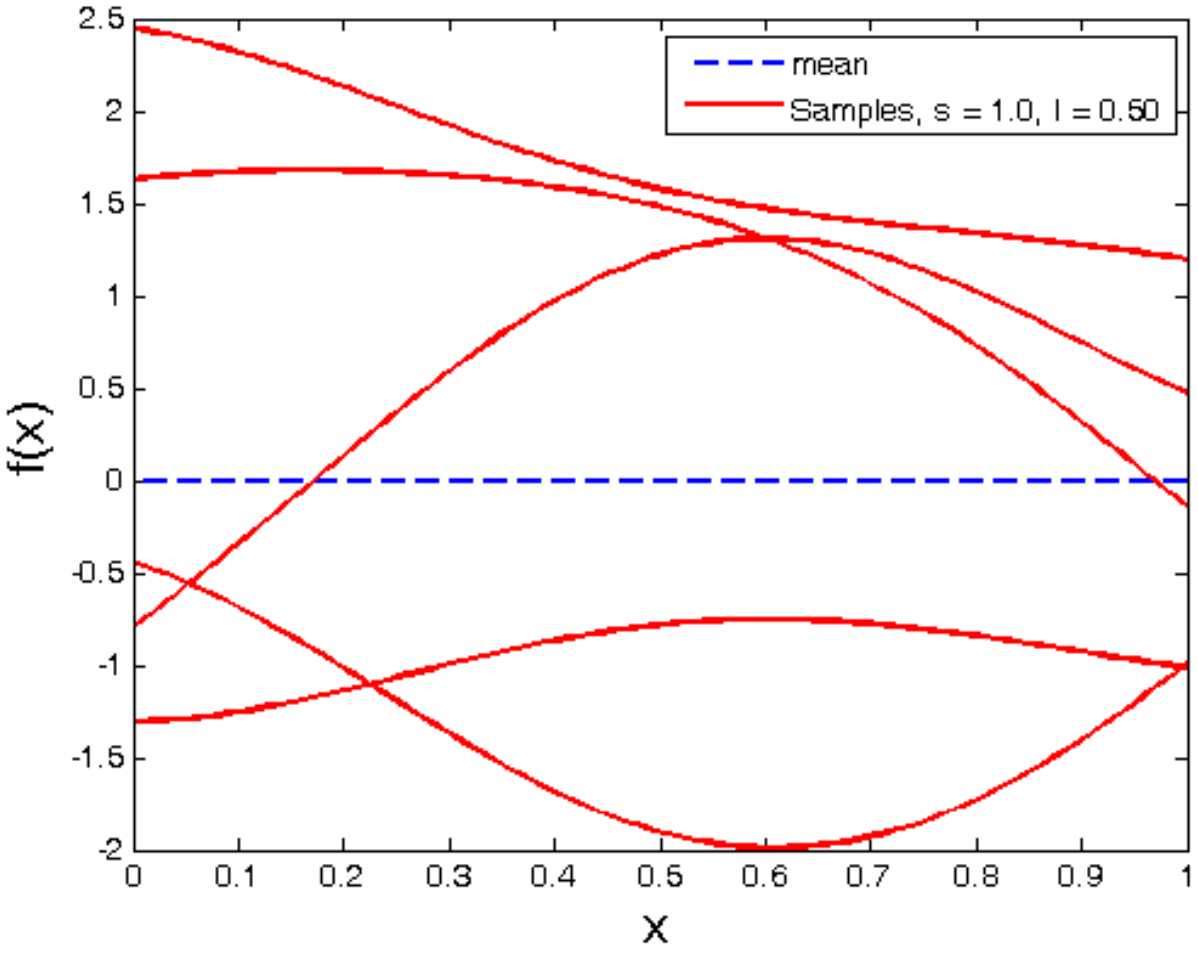
Small length scale, zero mean

Sample from GP (1D)



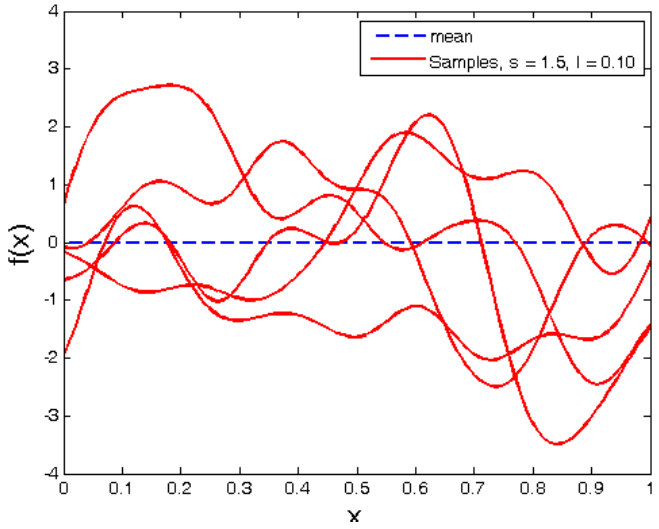
Medium length scale, zero mean

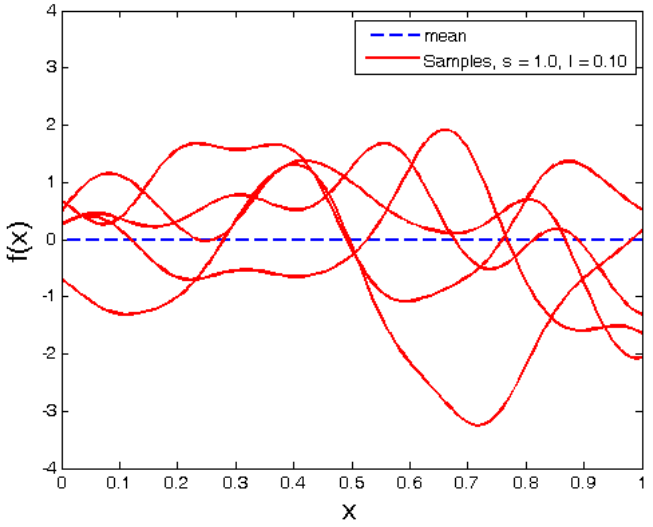
Sample from GP (1D)

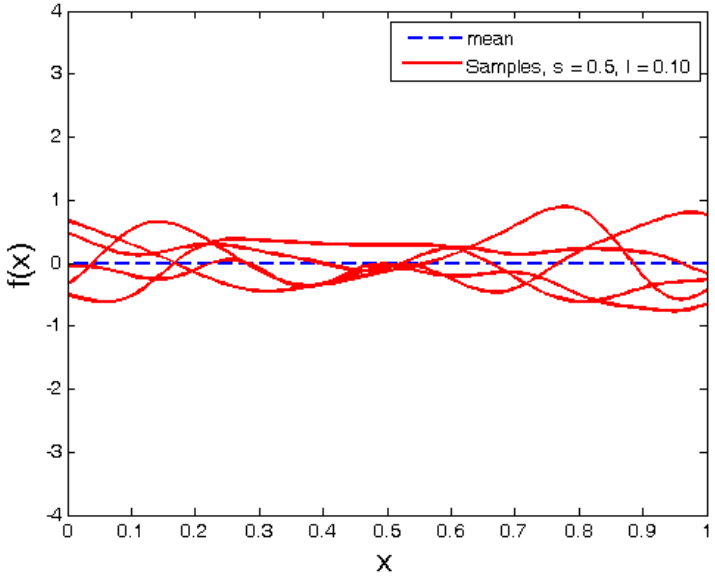


Large length scale, zero mean

Sample from GP (1D)



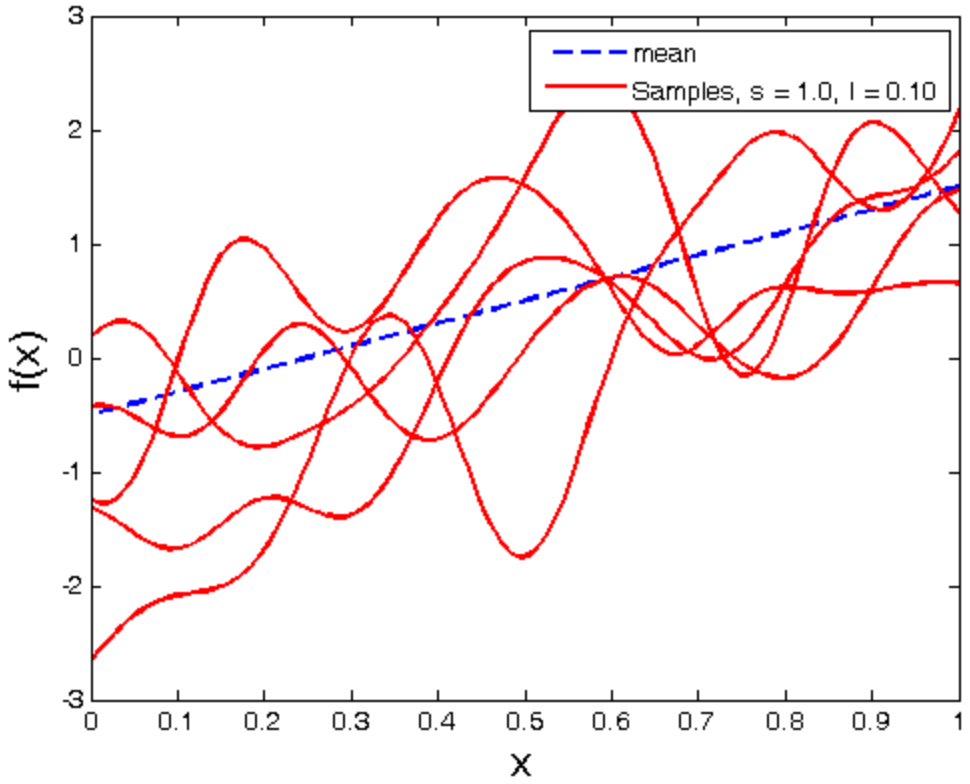




The effect of the signal strength.

The smaller it is, the less the prior samples vary.

Sample from GP (1D)

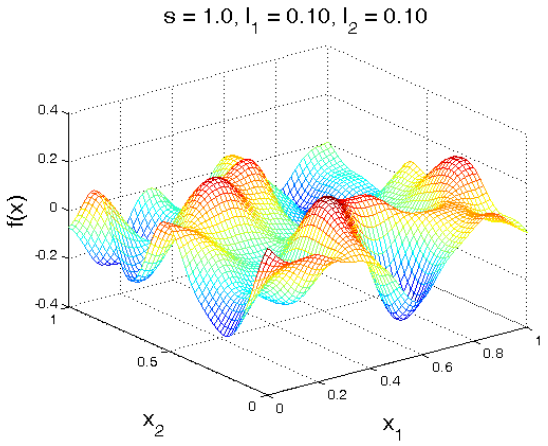


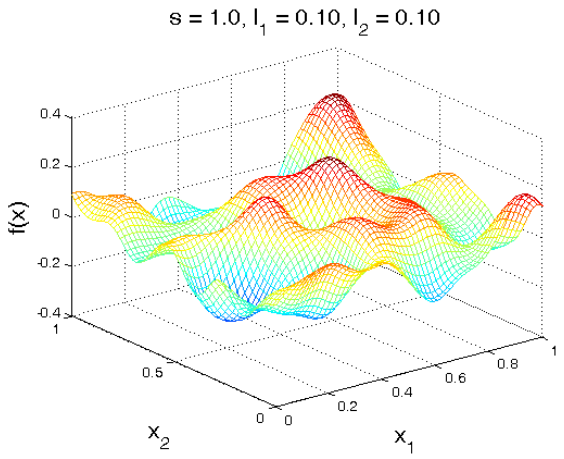
Using a linear plus constant mean.

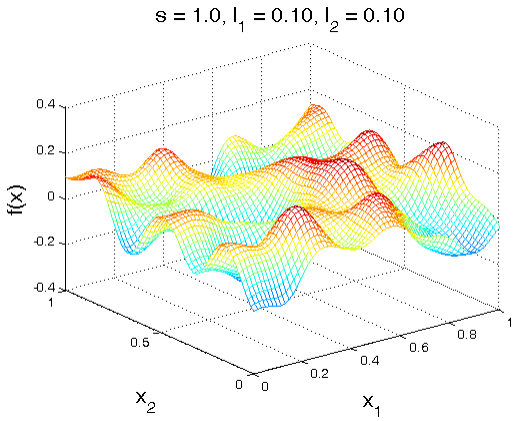
Matlab code: take_gp_samples_1d_non_zero_mean.m

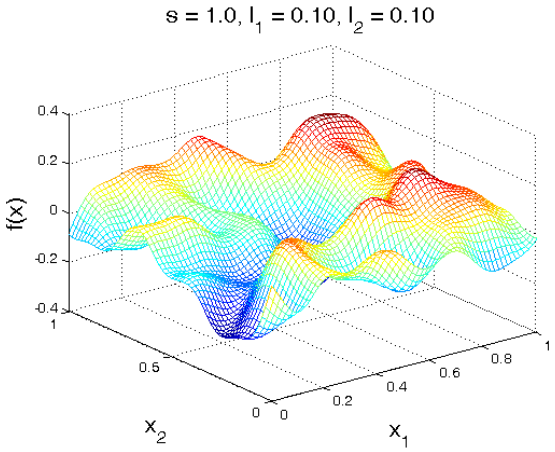
Purpose: Learn how to use a non-zero mean.

Sample from GP (2D)









Matlab code:
take_gp_samples_2d.m

Gaussian Process Regression

- To use Gaussian processes for regression, we need to model noise:

$$t_n = y_n + \varepsilon_n, \quad \text{with} \quad y_n = y(\boldsymbol{x}_n)$$

- For noise processes with a Gaussian distribution we obtain

$$p(t_n \mid y_n) = \mathcal{N}\left(t_n \mid y_n, \beta^{-1}\right)$$

- The joint distribution for $\boldsymbol{t} = (t_1, t_2, ..., t_n)^T$ conditioned on $\boldsymbol{y} = (y_1, y_2, ..., y_n)^T$ is then (since the noise is assumed to be independent for each data point)

$$p(\boldsymbol{t} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{t} \mid \boldsymbol{y}, \beta^{-1}\boldsymbol{I}_N\right)$$

Marginal Likelihood

- From the definition of a Gaussian process we have

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{0}, \boldsymbol{K})$$

- The kernel function that determines **K** is chosen such that for points $\boldsymbol{x}_n, \boldsymbol{x}_m$ that are similar the corresponding values $y(\boldsymbol{x}_n), y(\boldsymbol{x}_m)$ will be stronger correlated.

- For the marginal distribution $p(\boldsymbol{t})$, we need to integrate over $\boldsymbol{y}$ (see Appendix and earlier lecture notes):

$$p(\boldsymbol{t}) = \int p(\boldsymbol{t} \mid \boldsymbol{y})\, p(\boldsymbol{y})\, d\boldsymbol{y} = \mathcal{N}(\boldsymbol{t} \mid 0, \boldsymbol{C})$$

with $$\boldsymbol{C} = \boldsymbol{K} + \beta^{-1}\boldsymbol{I} \; or \; C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}$$

Appendix: Linear Gaussian Model

- Consider a linear Gaussian model: A Gaussian marginal distribution $p(\boldsymbol{x})$ and a Gaussian conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ in which $p(\boldsymbol{y}|\boldsymbol{x})$ has a mean that is a linear function of $\boldsymbol{x}$, and a covariance which is independent of $\boldsymbol{x}$.

$$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}, \Lambda^{-1}\right)$$

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}\right)$$

- We want to find $p(\boldsymbol{y})$. We start with the joint distribution over $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ which is a Gaussian and then integrate $\boldsymbol{x}$ out.

- Finally we obtained:

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\Lambda^{-1}\mathbf{A}^{T}\right)$$

Practical Kernel Function

- One widely used kernel function for Gaussian process regression is given here:

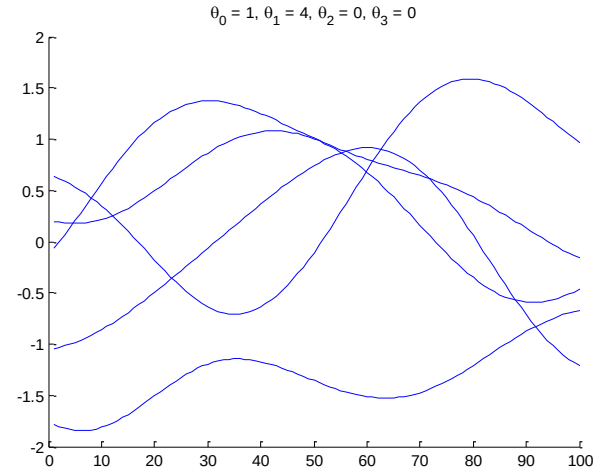
$$k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \theta_0 \exp\left(-\frac{\theta_1}{2}\left\|\boldsymbol{x}_n - \boldsymbol{x}_m\right\|^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

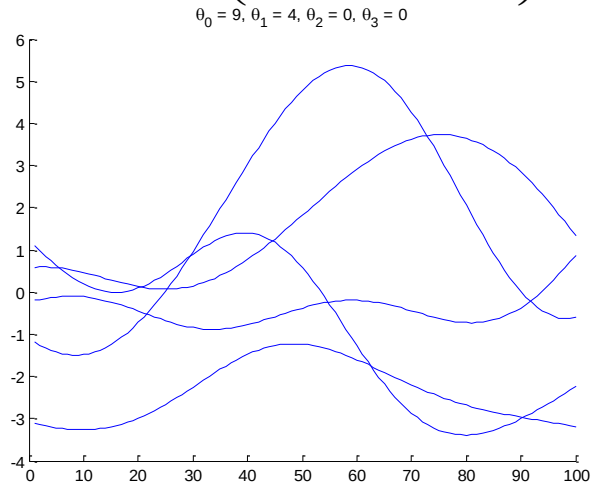
- The term involving θ_3 corresponds to a parametric model that is a linear function of the input variables.

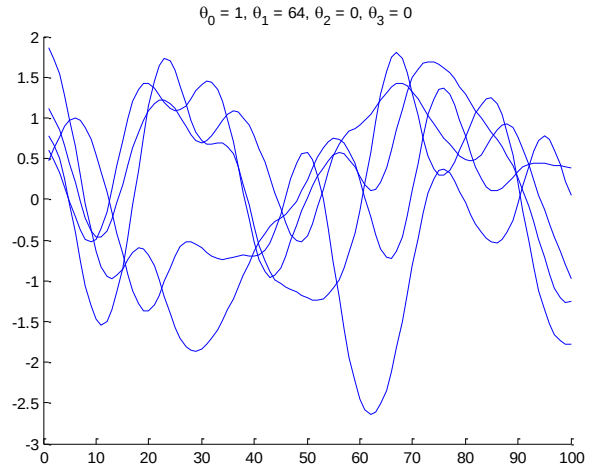
Distribution over Functions

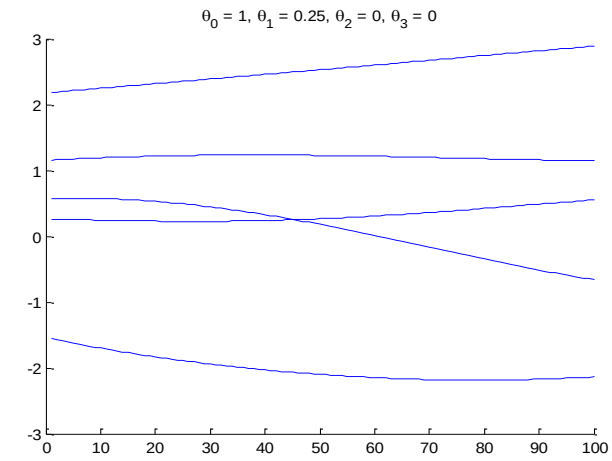
- Sample functions from prior $p(y)$ with common kernel function

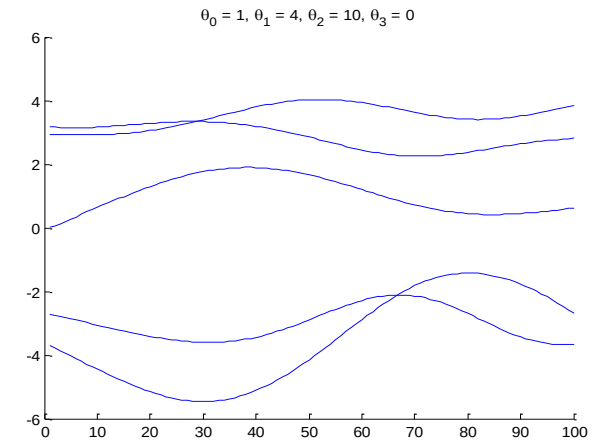
$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}\|\boldsymbol{x}_n - \boldsymbol{x}_m\|^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

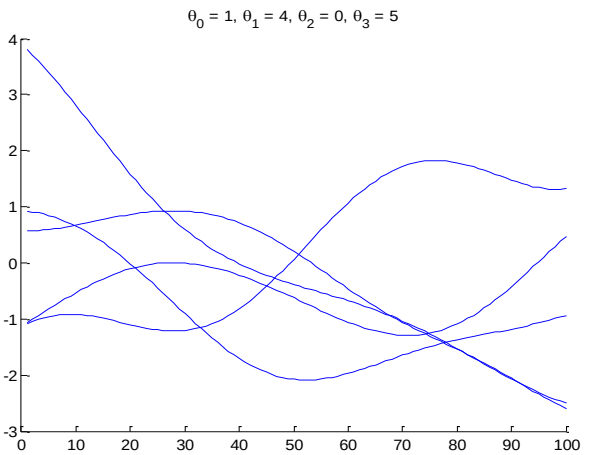












MatLab Code

Distribution Over Functions

- Sampling of data points $\{t_n\}$ from a Gaussian process.

- The blue curve shows a sample function from the Gaussian process prior over functions.

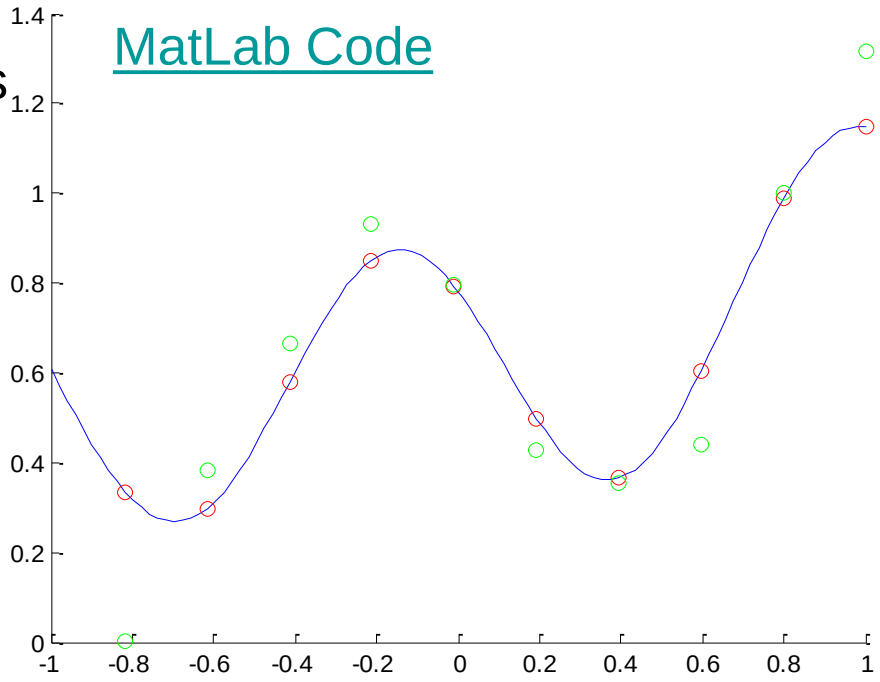
$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid \mathbf{0}, \mathbf{K})$$

- The red points show the values of y_n by evaluating the function at a set of input values $\{x_n\}$.

- The corresponding values of $\{t_n\}$, shown in green, are obtained by adding independent Gaussian noise to each of the $\{y_n\}$.

$$p(\mathbf{t}) = \mathcal{N}(\mathbf{t} \mid 0, \mathbf{C})$$

MatLab Code



Making Predictions

- So far we have a model for the joint probability distribution over sets of data points.
- For predictions t_{N+1} for a new input variable x_{N+1}, we need to evaluate the predictive distribution $p(t_{N+1}|\boldsymbol{t}_N)$.
- By partitioning the joint Gaussian distribution over $x_1, \ldots, x_N, x_{N+1}$, we obtain $p(t_{N+1}|\boldsymbol{t}_N)$ given by its mean and covariance (see following Appendix with results for linear Gaussian models discussed in an earlier lecture).
- The joint distribution over $t_1, \ldots, t_N, t_{N+1}$ is:

$$p(\boldsymbol{t}_{N+1}) = \mathcal{N}(\boldsymbol{t}_{N+1} \mid \boldsymbol{0}, \boldsymbol{C}_{N+1})$$

- We partition the covariance matrix as follows:

Making Predictions

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1} \mid \mathbf{0}, \mathbf{C}_{N+1})$$

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix}$$

- Here $\mathbf{C}_N$ is the $N \times N$ covariance matrix with elements given by

$$\mathbf{C}_N(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}, \; n, m = 1, \ldots, N$$

- The vector $\mathbf{k}$ has elements $k(\mathbf{x}_n, \mathbf{x}_{N+1})$ for $n = 1, \ldots, N$, and the scalar

$$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$$

Appendix: Conditional Gaussian Distributions

- If two sets of variables are jointly Gaussian, then the conditional distribution of one set conditioned on the other is again Gaussian.

- Suppose $\boldsymbol{x}$ is a $\boldsymbol{D}$-dimensional vector with Gaussian distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and that we partition $\boldsymbol{x}$ into two disjoint subsets $\boldsymbol{x_a}$ (M components) and $\boldsymbol{x_b}$ ($D - M$ components).

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{pmatrix}$$

Appendix: Conditional Gaussian Distributions

- This partition also implies similar partitions for the mean and covariance.

$$\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$$

- $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$ implies that $\boldsymbol{\Sigma_{aa}}$ ***and*** $\boldsymbol{\Sigma_{bb}}$ are symmetric and

$$\Sigma_{ba} = \Sigma_{ab}^T$$

Appendix: The Precision Matrix

- We define the precision matrix Λ as Σ^{-1}.

- Its partition is given as above

$$\Lambda = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}$$

where from $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$ we conclude that $\boldsymbol{\Lambda_{aa}}$ and $\boldsymbol{\Lambda_{bb}}$ are symmetric and

$$\Lambda_{ba} = \Lambda_{ab}^T$$

- Note that the above partition does NOT imply that $\boldsymbol{\Lambda_{aa}}$ is the inverse of $\boldsymbol{\Sigma_{aa}}$, etc.

Appendix: The Conditional Distribution

- We are now interested to compute $p(\boldsymbol{x}_a|\boldsymbol{x}_b)$. An easy way to do that is to look at $p(\boldsymbol{x}_a, \boldsymbol{x}_b)$ considering $\boldsymbol{x}_b$ constant.

- We have proved that the conditional is Gaussian with mean and variance given as:

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \Lambda_{aa}^{-1}\Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b) = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$$

$$\Sigma_{a|b} = \Lambda_{aa}^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$$

- Note that the conditional mean is linear in $\boldsymbol{x}_b$ and the conditional variance is independent of $\boldsymbol{x}_b$.

Making Predictions

- Using the results from the Appendix, $p(t_{N+1}|\boldsymbol{t})$ is a Gaussian with mean and covariance:

$$\boldsymbol{t} = \left(t_1, \ldots, t_N, t_{N+1}\right)^T = (b \quad a)^T \qquad \boldsymbol{C}_{N+1} = \begin{pmatrix} \boldsymbol{C}_N & \boldsymbol{k} \\ \boldsymbol{k}^T & c \end{pmatrix} \qquad p\left(\boldsymbol{t}_{N+1}\right) = \mathcal{N}\left(\boldsymbol{t}_{N+1} \,|\, \boldsymbol{0}, \boldsymbol{C}_{N+1}\right)$$

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \Rightarrow \qquad \boldsymbol{\mu}_{t_{N+1}|\boldsymbol{t}_N} = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N$$

$$\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \Rightarrow \boldsymbol{\Sigma}_{t_{N+1}|\boldsymbol{t}_N} = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}$$

- Finally we obtain $p(t_{N+1}|\boldsymbol{t})$ with both the mean and variance functions of $\boldsymbol{x}_{N+1}$

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1}), \quad a_n = \left(\boldsymbol{C}_N^{-1} \boldsymbol{t}_N\right)_n,$$

$$\boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

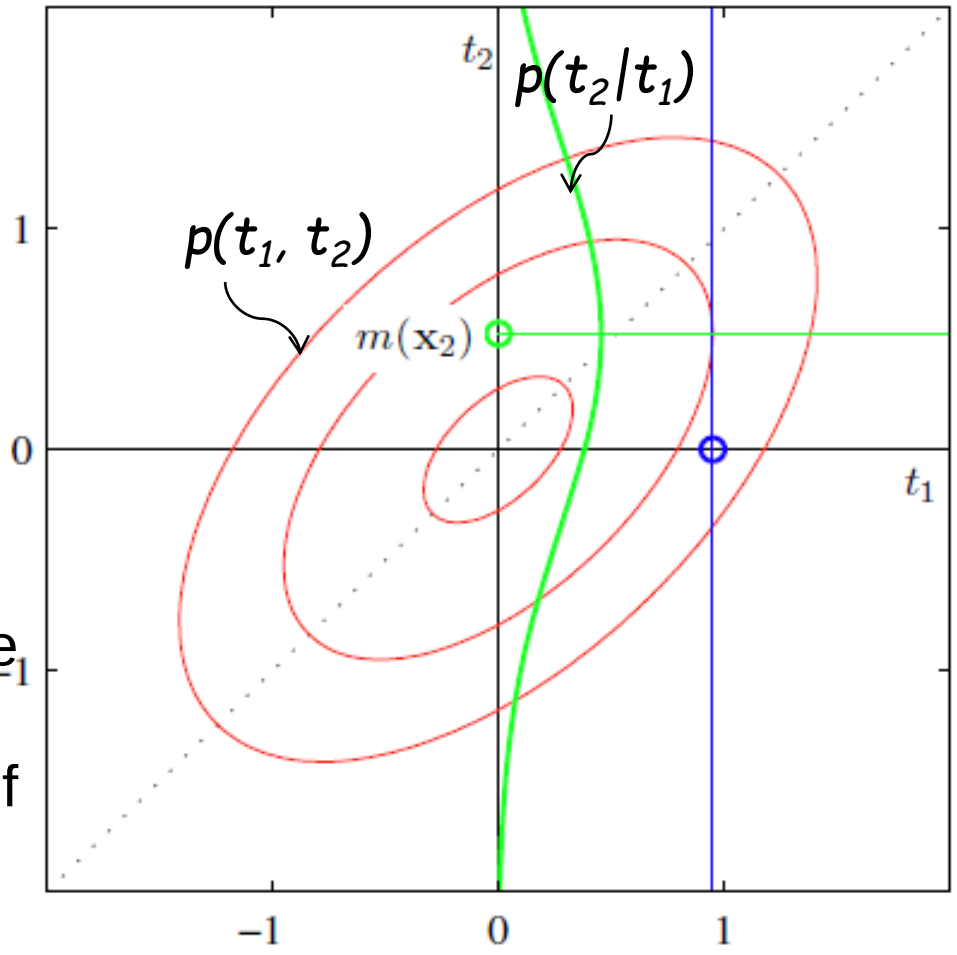
- If $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ depends only on $||\boldsymbol{x}_n - \boldsymbol{x}_m||$, then we obtain an expansion in radial basis functions.

Gaussian Process Regression

- Illustration of the mechanism of Gaussian process regression for the case of one training point and one test point.

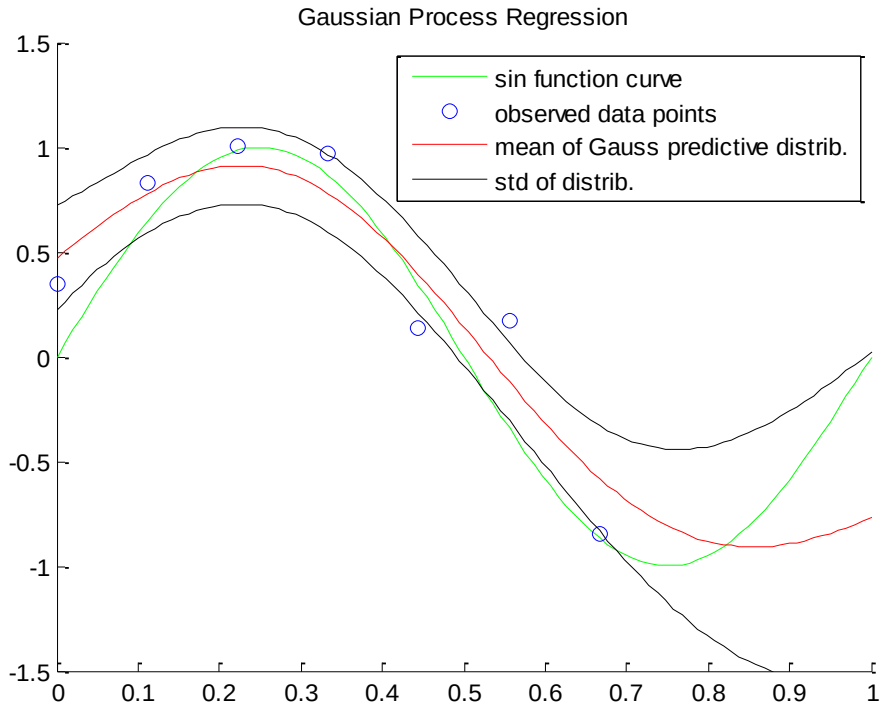
- The red ellipses show contours of the joint distribution $p(t_1, t_2)$.

- t_1 *is the* training data point, and conditioning on the value of t_1 *(*blue line), we obtain $p(t_2|t_1)$ shown as a function of t_2 by the green curve.



Making Predictions

- Green curve: original sinusoidal function; blue points: sampled training data points with additional noise; red line: mean estimate; black lines: +/- 2σ.
- Note the increase of uncertainty in the right of the data points.



MatLab Code

Gaussian Process Regression

- Assume we observe some $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ and $\boldsymbol{t} = (t_1, \dots, t_n)$.
- We are looking for the function $y = f(\boldsymbol{x})$.
- We pick a **likelihood** for the observations, e.g. Gaussian:

$$t|x, f(\cdot), \sigma^2 \sim \mathcal{N}(f(x), \sigma^2).$$

- We pick a **prior** for $f(\cdot)$. Because it is a function we pick a GP (zero mean for simplicity):

$$f(\cdot)|\theta \sim GP\big(m(\cdot), k(\cdot,\cdot)\big).$$

where θ collectively denotes the parameters of the mean and the covariance function (e.g. length scales, signal strength etc.).

Gaussian Process Regression

Remember that the GP really means:

$$\big(f(\boldsymbol{x}_1),\dots,f(\boldsymbol{x}_n)\big)|\theta \sim \mathcal{N}_n\left(\begin{pmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_n) \end{pmatrix}, \begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n,\boldsymbol{x}_n) \end{pmatrix}\right).$$

Therefore, the **likelihood** of the data is:

$$\boldsymbol{t}|\boldsymbol{X},\theta,\sigma^2 = \mathcal{N}_n\left(\begin{pmatrix} m(\boldsymbol{x}_1) \\ \vdots \\ m(\boldsymbol{x}_n) \end{pmatrix}, \begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1)+\sigma^2 & \cdots & k(\boldsymbol{x}_1,\boldsymbol{x}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n,\boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n,\boldsymbol{x}_n)+\sigma^2 \end{pmatrix}\right)$$
$$= \mathcal{N}_{\boldsymbol{n}}(\boldsymbol{m}(\boldsymbol{X}), k(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I}_n).$$

Gaussian Process Regression

Now, assume you want to make predictions at $\boldsymbol{x}^*$. Let us write:

$$f^* = f(\boldsymbol{x}^*).$$

(Remember that this is really a random variable because f is random.

Using the likelihood of the data and the definition of the GP, we have:

$$\begin{pmatrix} \boldsymbol{t} \\ f^* \end{pmatrix} | \boldsymbol{x}^*, \boldsymbol{X}, \theta, \sigma^2 \sim \mathcal{N}_n \left(\begin{pmatrix} \boldsymbol{m}(\boldsymbol{X}) \\ m(\boldsymbol{x}^*) \end{pmatrix}, \begin{pmatrix} k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n & k(\boldsymbol{X}, \boldsymbol{x}^*) \\ k(\boldsymbol{x}^*, \boldsymbol{X}) & k(\boldsymbol{x}^*, \boldsymbol{x}^*) \end{pmatrix} \right),$$

where

$$k(\boldsymbol{X}, \boldsymbol{x}^*) = (k(\boldsymbol{x}_1, \boldsymbol{x}^*) \quad \dots \quad k(\boldsymbol{x}_n, \boldsymbol{x}^*)),$$

and

$$k(\boldsymbol{x}^*, \boldsymbol{X}) = k(\boldsymbol{X}, \boldsymbol{x}^*)^T.$$

Now, all you have to do is find the probability density of f^*, conditioned on the rest:

$$f^* | \boldsymbol{x}^*, \boldsymbol{X}, \boldsymbol{t}, \theta, \sigma^2 \sim \mathcal{N}\left(m^*(\boldsymbol{x}^*), k^*(\boldsymbol{x}^*, \boldsymbol{x}^*)\right),$$

where

$$m^*(\boldsymbol{x}^*) = m(\boldsymbol{x}^*) + k(\boldsymbol{x}^*, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1}\left(\boldsymbol{y} - \boldsymbol{m}(\boldsymbol{X})\right),$$

$$k^*(\boldsymbol{x}^*, \boldsymbol{x}^*) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(\boldsymbol{x}^*, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1} k(\boldsymbol{X}, \boldsymbol{x}^*).$$

Training the Model

The model may be trained by maximizing the **marginal likelihood**:

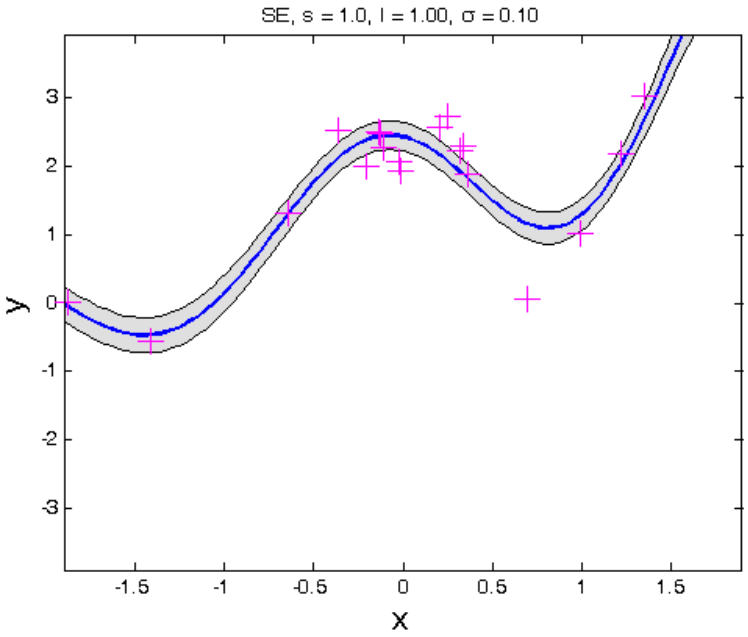
$$\boldsymbol{t}|\boldsymbol{X},\theta,\sigma^2 \sim \mathcal{N}_{\boldsymbol{n}}(\boldsymbol{m}(\boldsymbol{X}),k(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I}_n).$$

(Marginal because $f(\cdot)$ has been integrated out.)

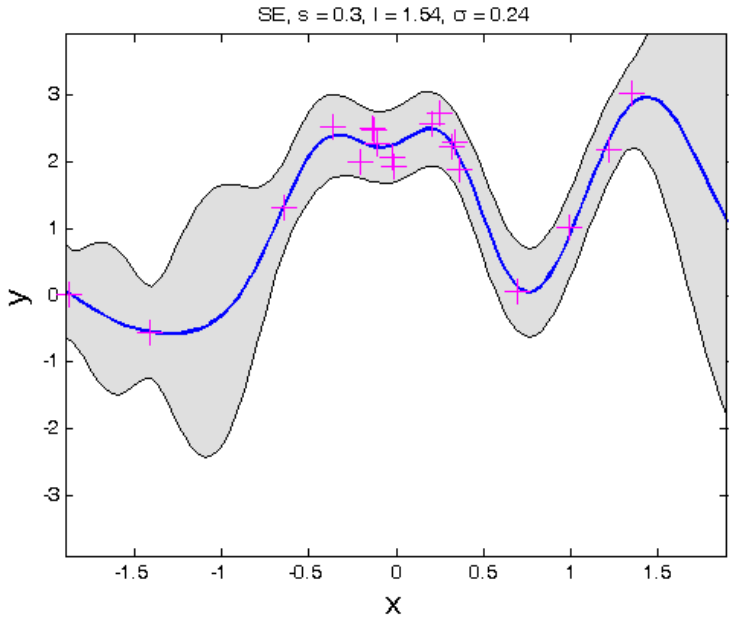
Usually, we maximize the logarithm:

$$\mathcal{L}(\theta,\sigma^2)=\log p(\boldsymbol{t}|\boldsymbol{X},\theta,\sigma^2)$$
$$=-\frac{1}{2}\big(\boldsymbol{t}-\boldsymbol{m}(\boldsymbol{X})\big)^T(k(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I}_n)^{-1}\big(\boldsymbol{t}-\boldsymbol{m}(\boldsymbol{X})\big)$$
$$-\frac{1}{2}\log|k(\boldsymbol{X},\boldsymbol{X})|-\frac{n}{2}\log 2\pi\,.$$

Example



Initial fit



Fit after maximizing the marginal likelihood.

Matlab code: gp_example_1.m

Joint Predictions

- Notice that similar equations hold if you want to write down the joint conditional predictive density for:

$$\boldsymbol{X}^* = (\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_{n^*}^*).$$

Specifically:

$$f^* | \boldsymbol{X}^*, \boldsymbol{X}, \boldsymbol{t}, \theta, \sigma^2 \sim \mathcal{N}\left(\boldsymbol{m}^*(\boldsymbol{X}^*), k^*(\boldsymbol{X}^*, \boldsymbol{X}^*)\right),$$

where

$$\boldsymbol{m}^*(\boldsymbol{X}^*) = \boldsymbol{m}(\boldsymbol{X}^*) + k(\boldsymbol{X}^*, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1}(\boldsymbol{t} - \boldsymbol{m}(\boldsymbol{X})),$$

$$k^*(\boldsymbol{X}^*, \boldsymbol{X}^*) = k(\boldsymbol{X}^*, \boldsymbol{X}^*) - k(\boldsymbol{X}^*, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1} k(\boldsymbol{X}, \boldsymbol{X}^*).$$

Posterior GP

- Invoking Kolmogorov's theorem on the construction of random fields, our model for the data is another GP (posterior GP), defined by:

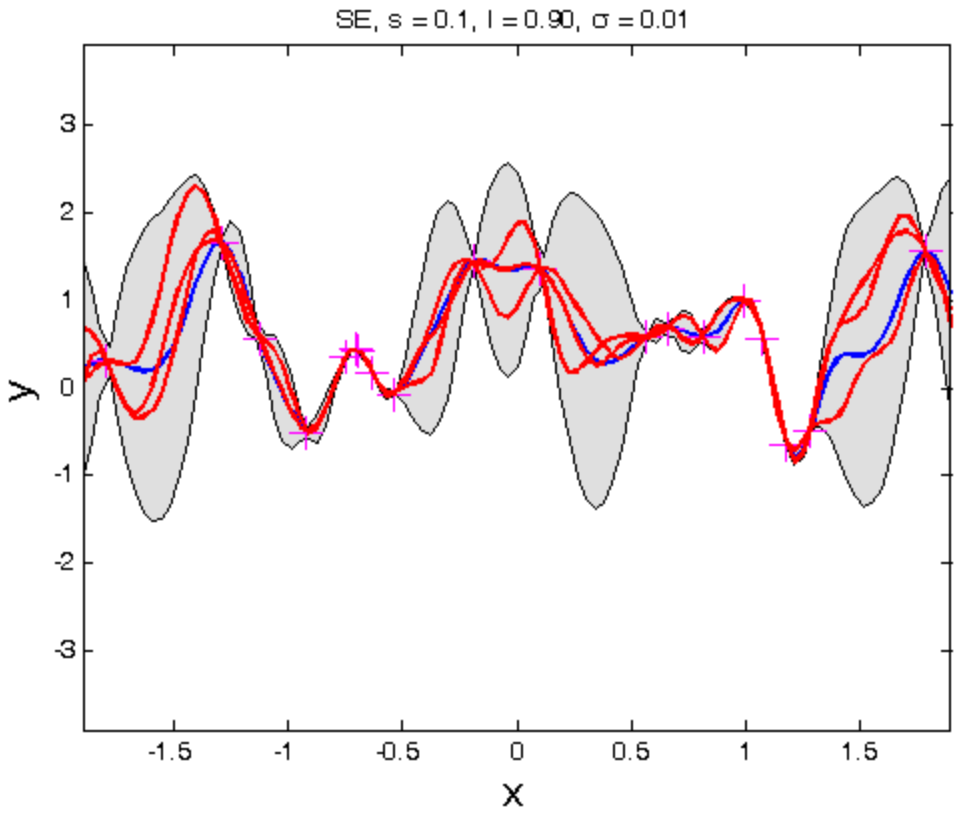
$$f(\cdot)|\boldsymbol{X}, \boldsymbol{t}, \theta, \sigma^2 \sim GP\left(m^*(\cdot), k^*(\cdot,\cdot)\right),$$

where

$$m^*(\boldsymbol{x}) = m(\boldsymbol{x}) + k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1}(\boldsymbol{t} - \boldsymbol{m}(\boldsymbol{X})),$$

$$k^*(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \boldsymbol{X})(k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n)^{-1} k(\boldsymbol{X}, \boldsymbol{x}').$$

Samples from the Posterior GP



Samples from the posterior GP.

Matlab code: gp_example_2.m

Restrictions on the kernel Matrix

- The covariance matrix must be positive definite.

$$C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}, n, m = 1, \ldots, N$$

- If λ_i is an eigenvalue of $\boldsymbol{K}$, then the corresponding eigenvalue of $\boldsymbol{C}$ will be $\lambda_i + \beta^{-1}$.

- It is therefore sufficient that the kernel matrix $k(\boldsymbol{x_n}, \boldsymbol{x_m})$ be positive semidefinite for any pair of points $\boldsymbol{x}_n$ and $\boldsymbol{x_m}$, so that $\lambda_i > 0$. Any $\lambda_i = 0$ will still give rise to a positive eigenvalue for $\boldsymbol{C}$ because $\beta > 0$.

- This is the same restriction on the kernel function discussed in an earlier lecture, and so we can exploit the techniques discussed there.

GPs Vs. Linear Regression Models

- Note that the mean of the conditional $p(t_{N+1}|\boldsymbol{t})$

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), ..., k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

can be written as:

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1}),$$

where

$$\boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \left(a_1 ... a_N\right)^T$$

- Thus if the kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ depends only on the distance $\|\boldsymbol{x}_n - \boldsymbol{x}_m\|$ we obtain an expansion in radial basis.

GPs Vs. Linear Regression Models

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), ..., k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

- These results define the predictive distribution for Gaussian process regression with an arbitrary kernel function $k(\boldsymbol{x}_n, \boldsymbol{x}_m)$

- If the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ is defined in terms of a finite set of basis functions, the results become identical to those we discussed in an earlier lectures on regression (see proof in the next two slides)

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \int p(t \mid x, \boldsymbol{w}) p(\boldsymbol{w} \mid \boldsymbol{x}, \boldsymbol{t}) d\boldsymbol{w} = \mathcal{N}\left(t \mid \boldsymbol{m}_N^T \boldsymbol{\phi}(x), \sigma_N^2(x)\right)$$

$$\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$$

$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

$$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T S_N \boldsymbol{\phi}(x)$$

Kernels Defined by a Finite Basis

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T$$

- We can write the vector $\boldsymbol{k}$ as:

$$\boldsymbol{k} = \left(k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1})\right)^T = \frac{1}{\alpha} \begin{pmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_N) \end{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) = \frac{1}{\alpha} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})$$

and use

$$\boldsymbol{C}_N = \alpha^{-1} \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N$$

- Using this, the predicted mean takes the form

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N = \frac{1}{\alpha} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \boldsymbol{\Phi}^T \left(\alpha^{-1} \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N\right)^{-1} \boldsymbol{t}_N =$$

$$\alpha^{-1} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \underset{\boldsymbol{A}}{\beta \boldsymbol{\Phi}^T} \left(\underset{\boldsymbol{B}}{\alpha^{-1} \boldsymbol{\Phi}} \underset{\boldsymbol{A}}{\beta \boldsymbol{\Phi}^T} + \boldsymbol{I}_N \right)^{-1} \boldsymbol{t}_N \underset{(\boldsymbol{I}+\boldsymbol{AB})^{-1}\boldsymbol{A}=\boldsymbol{A}(\boldsymbol{I}+\boldsymbol{BA})^{-1}}{=} \alpha^{-1} \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \left(\beta \boldsymbol{\Phi}^T \alpha^{-1} \boldsymbol{\Phi} + \boldsymbol{I}_M\right)^{-1} \beta \boldsymbol{\Phi}^T \boldsymbol{t}_N =$$

$$\boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \beta \left(\beta \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \alpha \boldsymbol{I}_M\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}_N = \beta \boldsymbol{\phi}^T(\boldsymbol{x}_{N+1}) \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}_N, \text{ the required result}$$

Kernels Defined by a Finite Basis

$$\sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

- Using $\boldsymbol{k} = \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})$, $c = \alpha^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) + \beta^{-1}$ and $\boldsymbol{C}_N = \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N$, we compute the predictive variance as:

$$\sigma^2(\boldsymbol{x}_{N+1}) = \alpha^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) + \beta^{-1} - \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \boldsymbol{\Phi}^T \alpha^{-1} \left(\alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta^{-1} \boldsymbol{I}_N \right)^{-1} \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) =$$

$$\beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T \left\{ \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} - \underbrace{\boldsymbol{\Phi}^T}_{\boldsymbol{B}} \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} \left(\underbrace{\alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T}_{\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}} + \underbrace{\beta^{-1} \boldsymbol{I}_N}_{\boldsymbol{D}} \right)^{-1} \underbrace{\boldsymbol{\Phi}}_{\boldsymbol{C}} \underbrace{\alpha^{-1} \boldsymbol{I}_M}_{\boldsymbol{A}^{-1}} \right\} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) =$$

$$\beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T (\alpha \boldsymbol{I}_M + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\phi}(\boldsymbol{x}_{N+1}) = \beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}_{N+1})^T S_N \boldsymbol{\phi}(\boldsymbol{x}_{N+1}),$$ the required result

- Here we used the Woodbury identity with $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ as shown.

$$(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} + \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1}$$

Gaussian process Vs. Basis Function Model

$$m(\boldsymbol{x}_{N+1}) = \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{t}_N, \quad \sigma^2(\boldsymbol{x}_{N+1}) = c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}, \quad c = k(\boldsymbol{x}_{N+1}, \boldsymbol{x}_{N+1}) + \beta^{-1}$$

$$p(t \mid x, \boldsymbol{x}, \boldsymbol{t}) = \mathcal{N}\left(t \mid \boldsymbol{m}_N^T \boldsymbol{\phi}(x), \sigma_N^2(x)\right)$$

$$\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^T \boldsymbol{t}$$
$$\boldsymbol{S}_N^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}$$

$$\sigma_N^2(x) = \frac{1}{\beta} + \boldsymbol{\phi}(x)^T S_N \boldsymbol{\phi}(x)$$

- In Gaussian processes, we invert $\boldsymbol{C}_N$ of size $N \times N$, cost $\mathcal{O}(N^3)$
- In the basis function model, we invert S_N, size M $\times M$, cost $\mathcal{O}(M^3)$
- These matrix inversions must be performed once for the given training set.
- For each new test point, both methods require a vector-matrix multiplications ($\mathcal{O}(N^2)$ vs. $\mathcal{O}(M^2)$). If $M < N$, it will be computationally more efficient to work in the basis function framework.
- An advantage of Gaussian processes is that we can consider covariance functions that can only be expressed in terms of an infinite number of basis functions.

Large Training Data Sets

- For large training data sets, GPs can become infeasible, and approximation schemes have been developed with better scaling with training set size than the exact approach.

- Gibbs, M. N. (1997). *Bayesian Gaussian processes for regression and classification*. Phd thesis, University of Cambridge.
- Tresp, V. (2001). Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery* **5**(3), 197–211.
- Smola, A. J. and P. Bartlett (2001). Sparse greedy Gaussian process regression. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 619–625. MIT Press.
- Williams, C. K. I. and M. Seeger (2001). Using the Nystrom method to speed up kernel machines. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 682–688. MIT Press.
- Csató, L. and M. Opper (2002). Sparse on-line Gaussian processes. *Neural Computation* **14**(3), 641– 668.
- Seeger, M., C. K. I. Williams, and N. Lawrence (2003). Fast forward selection to speed up sparse Gaussian processes. In C. M. Bishop and B. Frey (Eds.), *Proceedings Ninth International Workshop on Artificial Intelligence and Statistics, Key West, Florida*.
- Bishop, C. M. and I. T. Nabney (2008). *Pattern Recognition and Machine Learning: A Matlab Companion*. Springer. In preparation.

Extension of Gaussian Processes

- Extension of GPs to multiple target variables (co-kriging) is straightforward.
- Other extensions of GPs include
  - modelling the distribution over low-dimensional manifolds for unsupervised learning and
  - the solution of SPDEs

- Cressie, N. (1993). *Statistics for Spatial Data*.Wiley.
- Bishop, C. M., M. Svensén, and C. K. I. Williams (1998a). Developments of the Generative Topographic Mapping. *Neurocomputing* **21**, 203– 224.
- Graepel, T. (2003). Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 234–241.

Learning the Hyperparameters of GPs and Automatic Relevance Determination

Learning the Hyperparameters θ

- Instead of fixing the covariance function, use a parametric family of functions and infer the parameters from the data.
- Parameters $\boldsymbol{\theta}$: length scale of correlations, and precision of noise β.
- Techniques for learning $\boldsymbol{\theta}$ are based on the likelihood $p(\boldsymbol{t}|\boldsymbol{\theta})$ – analogous to model evidence techniques used earlier.
- Simplest approach:
  - Maximizing the log-likelihood (MLE): ln p(**t**| θ)

$$\ln p(\boldsymbol{t}\,|\,\boldsymbol{\theta}) = -\frac{1}{2}\ln|\boldsymbol{C}_N| - \frac{1}{2}\boldsymbol{t}^T\boldsymbol{C}_N^{-1}\boldsymbol{t} - \frac{N}{2}\ln(2\pi)$$

  - Use gradient optimization

$$\frac{\partial}{\partial\theta_i}\ln p(\boldsymbol{t}\,|\,\boldsymbol{\theta}) = -\frac{1}{2}tr\left[\boldsymbol{C}_N^{-1}\frac{\partial\boldsymbol{C}_N}{\partial\theta_i}\right] + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{C}_N^{-1}\frac{\partial\boldsymbol{C}_N}{\partial\theta_i}\boldsymbol{C}_N^{-1}\boldsymbol{t}$$

  - $\ln p(\boldsymbol{t}|\,\boldsymbol{\theta})$ is in general non-convex and can have multiple maxima

- Fletcher, R. (1987). *Practical Methods of Optimization* (Second ed.). Wiley.
- Nocedal, J. and S. J. Wright (1999). *Numerical Optimization*. Springer.

Appendix

- Show that

$$\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$$

Indeed note that

$$\frac{\partial}{\partial x}\left(\boldsymbol{A}\boldsymbol{A}^{-1}\right) = \boldsymbol{0} \Rightarrow \frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1} + \boldsymbol{A}\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = 0 \Rightarrow \frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$$

- Show that ($\boldsymbol{A}$ symmetric)

$$\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$$

Indeed using the eigen-decomposition of $\boldsymbol{A}$, note that

$$\boldsymbol{A} = \sum_i \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^T,\ \boldsymbol{A}^{-1} = \sum_i \frac{1}{\lambda_i}\boldsymbol{u}_i \boldsymbol{u}_i^T,\ tr\boldsymbol{A} = \sum_i \lambda_i,\ |\boldsymbol{A}| = \prod_i \lambda_i \Rightarrow$$

$$\ln|\boldsymbol{A}| = \sum_i \ln\lambda_i \Rightarrow \frac{\partial}{\partial x}\ln|\boldsymbol{A}| = \sum_i \frac{\partial \ln\lambda_i}{\partial x} = \sum_i \frac{1}{\lambda_i}\frac{\partial \lambda_i}{\partial x} = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$$

Learning the Hyperparameters θ

- One can also introduce a prior $p(\boldsymbol{\theta})$ and maximize the log-posterior (maximum a posteriori): $\ln p(\boldsymbol{t}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})$
- In a Bayesian setting, we need to compute marginals over $\boldsymbol{\theta}$ weighted with $p(\boldsymbol{t}|\boldsymbol{\theta})p(\boldsymbol{\theta})$; this is not tractable → use approximations
- We have assumed that the contribution to the predictive variance arising from the noise β is constant. The noise might not be additive but dependent on x (heteroscedastic)

$$C(\boldsymbol{x}_n, \boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \beta^{-1}\delta_{nm}, n, m = 1, \ldots, N$$

- A second Gaussian process can be introduced to represent the dependency of β on $\boldsymbol{x}$.
  - Use a Gaussian process for $\ln \beta$ since $\beta \geq 0$.

- Goldberg, P. W., C. K. I. Williams, and C. M. Bishop (1998). Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems*, Volume 10, pp. 493–499. MIT Press.

Automatic Relevance Detection

- An additional hyperparameter can be introduced for each input dimension, e.g. in two-dimensions:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \eta_i \left(x_i - x_i^{'} \right)^2 \right)$$

- Hyperparameter optimization by maximum likelihood allows then a different weighting of each dimension.

- Unrelevant dimensions (with small weights) can be detected and discarded.

Automatic Relevance Detection

$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2}\sum_{i=1}^{2}\eta_i\left(x_i - x_i^{'}\right)^2\right)$$

- Samples from the resulting prior over functions y(**x**) are shown next for two settings of η_i.
- As a particular η_i becomes small, the function becomes insensitive to x_i.
- By adapting these parameters to a data set using ML, it becomes possible to detect input variables that have little effect on the predictive distribution (such inputs are discarded)

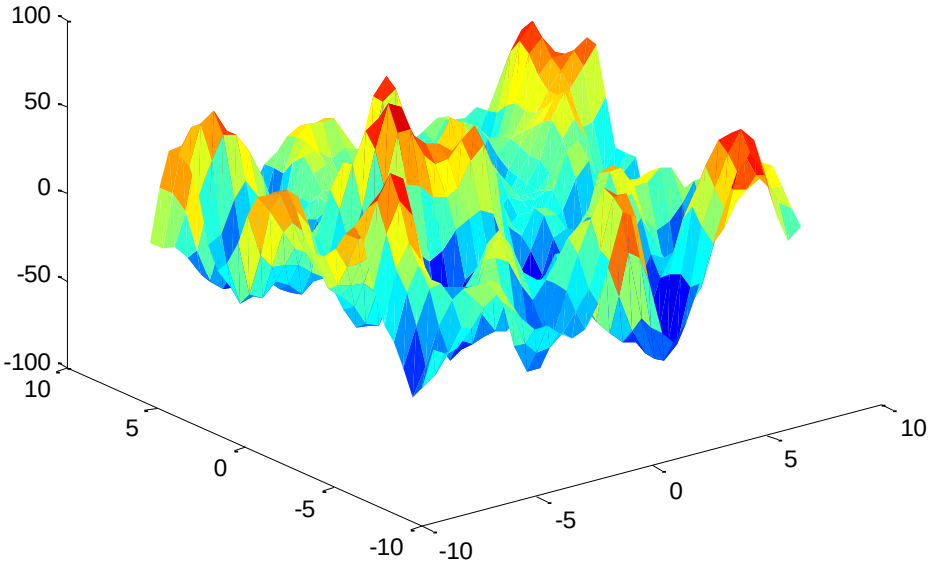
- MacKay, D. J. C. (1994). Bayesian methods for backprop networks. In E. Domany, J. L. van Hemmen, and K. Schulten (Eds.), *Models of Neural Networks, III*, Chapter 6, pp. 211–254. Springer.
- Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer. Lecture Notes in Statistics 118.

Automatic Relevance Detection

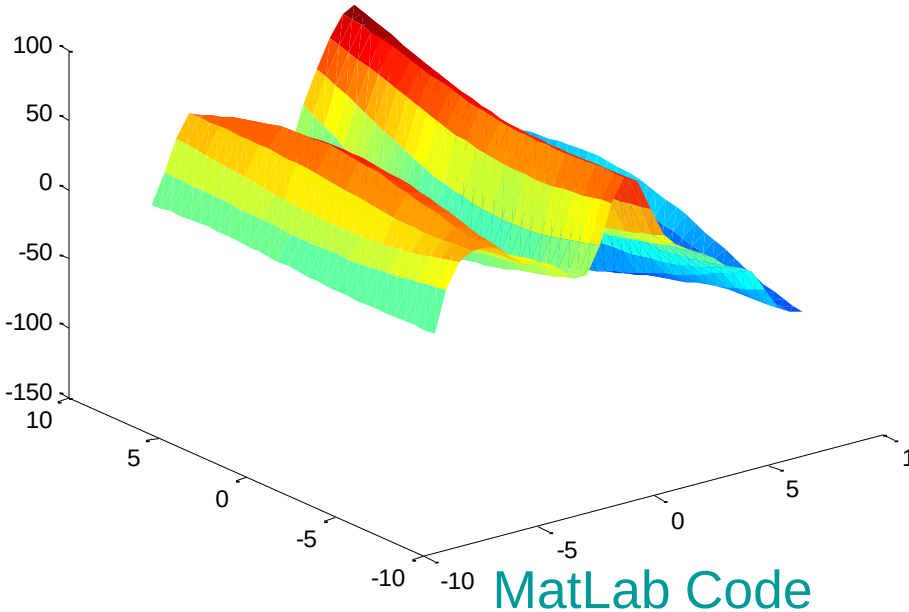
$$k(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp\left(-\frac{1}{2} \sum_{i=1}^{2} \eta_i \left(x_i - x_i' \right)^2 \right)$$

- Samples from the resulting prior over functions $y(\boldsymbol{x})$ are shown for two settings of η_i.

$\eta_1 = \eta_2 = 1$



$\eta_1 = 1, \eta_2 = 0.01$



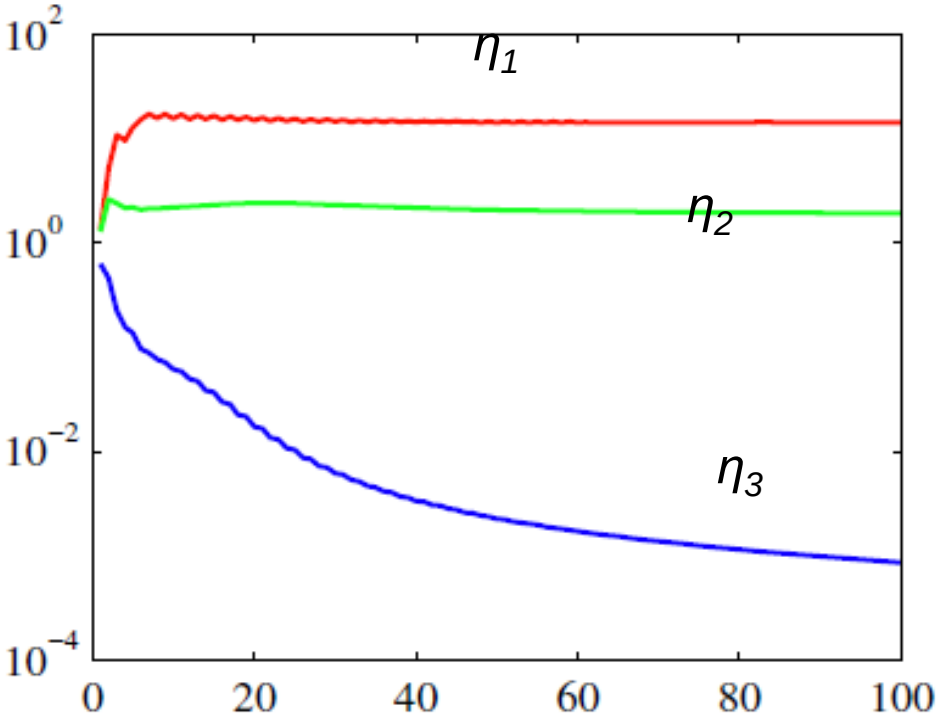
MatLab Code

Automatic Relevance Detection

- ARD is illustrated next using a data set having 3 inputs x_1, x_2 and x_3.
- The target variable t, is generated by sampling 100 values of x_1 from a Gaussian, evaluating the function $\sin(2\pi x_1)$, and then adding Gaussian noise.
- Values of x_2 are given by copying the corresponding values of x_1 and adding noise, and values of x_3 are sampled from an independent Gaussian distribution.
- Thus x_1 is a good predictor of t, x_2 is a more noisy predictor of t, and x_3 has only chance correlating with t.
- The marginal likelihood for a Gaussian process with ARD parameters η_1, η_2, η_3 is optimized using the scaled conjugate gradients algorithm.

Automatic Relevance Detection

- We see (log scale in the vertical axis) that η_1 converges to a relatively large value, η_2 converges to a much smaller value, and η_3 becomes very small indicating that x_3 is irrelevant for predicting t.



MatLab Code

- Nabney, I. T. (2002). *Netlab: Algorithms for Pattern Recognition*. Springer.

Automatic Relevance Detection

- The ARD framework can be applied to general kernels of the form:

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}\sum_{i=1}^{D}\eta_i\left(x_{ni} - x_{mi}\right)^2\right) + \theta_2 + \theta_3 \boldsymbol{x}_n^T \boldsymbol{x}_m$$

where D is the input dimensionality.

Applications of Gaussian Processes

- GP's started in geology. They are used to model geological beds.
- They are used in weather forecasting (that's how they get the contours you see every night on the news).
- GP's are very successful in modeling computer surrogates to be subsequently used in:
  - Inverse Problems (model calibration).
  - Uncertainty Quantification.
  - Optimization and Design.

The most important difficulties

- There are several difficulties with GPs:
  - **Large observation sets:** Remember, you have to find the Cholesky decomposition of a matrix as big as the number of observations.
  - **High-dimensions:** The meaning of distance in high-dimensions is problematic.
  - **Non-stationarity:** How do you represent functions whose length scales vary in the input space? How do you represent functions with several length scales?
- Research takes place mostly on these areas, as well as on applications.

That's all with GP's

- We have basically presented all the practical aspects of Gaussian Processes.
- That is all that is needed in order to use it in practice.
- For applications, the most important thing to understand is:

How does the choice of the covariance function affect the prior model?

- In particular:
  - How does the covariance affect the regularity assumption on the function?
  - What is the effect of the length scales?
  - What if the function is expected to have different length scales on different regions of space? (Non-stationarity).
  - How do I model periodicity?
  - Are there any good covariance functions for high-dimensions?
  - Are there covariance functions with compact support (dealing with high-dimensional data).

More on GP Kernels

More on Covariance Functions

- The most important ingredient of a GP.
- It encodes our assumptions about the function we wish to learn.
- It defines nearness.
- It defines similarity.

Definition: Covariance function.

*A symmetric function $k: \mathcal{X} \times \mathcal{X} \to R$ is called a **covariance function** if it is positive semi-definite. That is for any $x^{(i)} \in \mathcal{X}$ and $w_i \in R$, for $i = 1, \dots, n$, we have*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} w_i k\left(x^{(i)}, x^{(j)}\right) w_j \geq 0.$$

Stationarity and Isotropy

Definition: Stationary Covariance Function

*A covariance function is called **stationary** if*

$$k(x, x') = k(x - x').$$

Remarks:

1. A stationary covariance function is invariant to translations in the input space.
2. The Square Exponential is an example of a stationary covariance function.

Definition: Isotropic Covariance Function

*A covariance function is called **isotropic** if*

$$k(x, x') = k(|x - x'|).$$

Remarks:

1. An isotropic covariance function is invariant to translation and rotation in the input space.

Mean Square Continuity

Definition: Continuity in the Mean Square (MS) sense.

Let $x^{(1)}, x^{(2)}, \ldots$ be a sequence of points and x^* in $\mathcal{X}$ such that:

$$x^{(i)} \to x^*, \text{as } i \to \infty.$$

*Then a process $f(x)$ is **continuous in mean square** at x^* if*

$$\mathbb{E}\left[\left|f\left(x^{(i)}\right) - f(x^*)\right|^2\right] \to 0, \text{as } i \to \infty.$$

Theorem: *A Gaussian process is continuous in MS at x^* if and only if its covariance function $k(x, x')$ is continuous at $x = x' = x^*$.*

Proof: See Adler [1981, ch. 3].

Corollary: *For a stationary GP, it suffices that $k(x, x')$ is continuous at $x = x' = 0$*

Mean Square Differentiability

Definition: Mean Square differentiability.

Let $f(x)$ be a process and $x^* \in \mathcal{X}$. The process is called differentiable in mean square at x^* if the following limit exists:

$$\lim_{h \to 0} \mathbb{E}\left[\left|\frac{f(x^* + he_i) - f(x^*)}{h} - a\right|^2\right] = 0,$$

for some real number a.

If the process is MS differentiable at all $x \in \mathcal{X}$, a can be thought of as a random process representing the partial derivative of $f(x)$. Then we may write:

$$\lim_{h \to 0} \mathbb{E}\left[\left|\frac{f(x + he_i) - f(x)}{h} - \frac{\partial f(x)}{\partial x_i}\right|^2\right] = 0.$$

Theorem: *A GP is MS differentiable if and only if*

$$\frac{\partial^2 k(x, x')}{\partial x_i \partial x_i'}$$

exists and is finite.

Mean Square Differentiability

More generally…

Theorem: *If the 2r-th-order partial derivative*

$$\frac{\partial^{2r} k(x, x')}{\partial^2 x_{i_1} \dots \partial^2 x_{i_r}}$$

exists and is finite, then the r-th order partial derivative of the corresponding Gaussian process exists

$$\frac{\partial^r f(x)}{\partial x_{i_1} \dots \partial x_{i_r}}.$$

Remark: If the derivatives exists, then they are GP's! Their covariance is the corresponding derivative of the covariance function of the original GP!

Question: The previous remark can be used to learn a function using sampled values of its derivatives also. How?

Matern Covariance

We write it in one dimension:

$$k_{Matern}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|x - x'|}{\ell} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}|x - x'|}{\ell} \right),$$

where

- $\nu > 0$, modeling differentiability,
- $\ell > 0$, modeling length scale,
- $\Gamma(\nu)$, the Gamma function,
- K_{ν}, a modified Bessel function of the second kind.

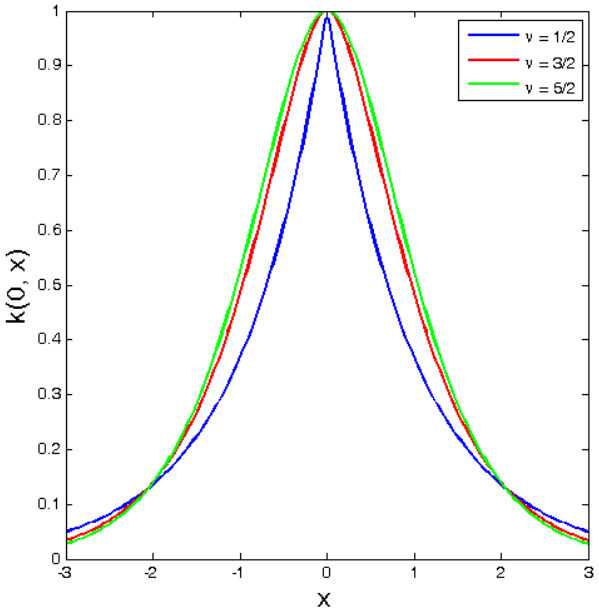
Properties:

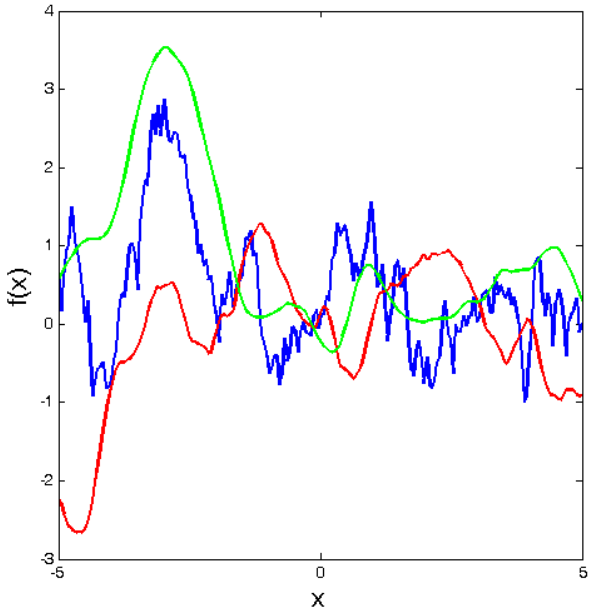
- As $\nu \to \infty$, it approaches the Square Exponential covariance.
- The process is k –times differentiable if and only if $\nu > k$.
- For $\nu = 1/2$, it gives the Exponential covariance:

$$k_{EXP}(x, x') = e^{-\frac{r}{\ell}},$$

which is MS continuous but not MS differentiable.

Matern Covariance





Matlab code: take_matern_samples.m

Rational Quadratic Covariance

Again, in one-dimension:

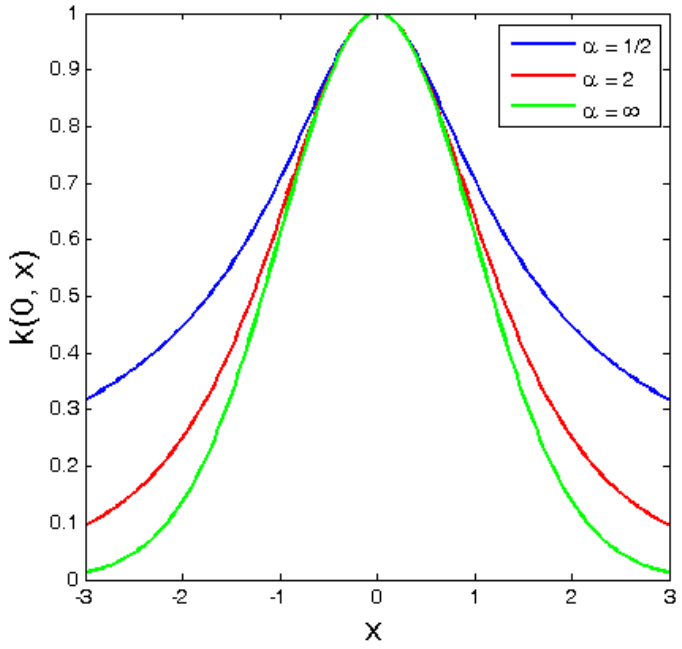
$$k_{RQ}(x, x') = \left(1 + \frac{r^2}{2a\ell}\right)^{-a},$$

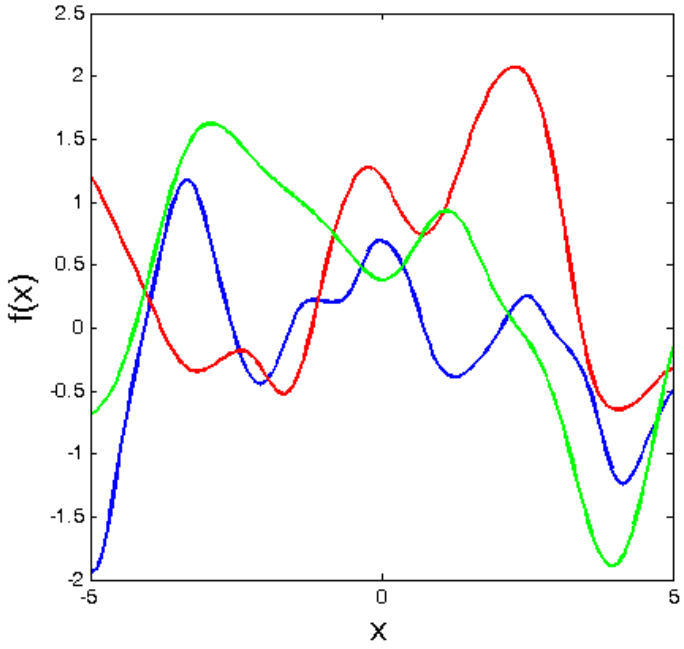
where $a > 0, \ell > 0$.

Properties:

- Infinitely differentiable.
- It can be thought of as a *scale mixture* of squared exponential covariance functions with different characteristic length-scales. See Rasmussen and Williams for the proof.
- As $a \to \infty$, it goes to a square exponential.

Rational Quadratic Covariance





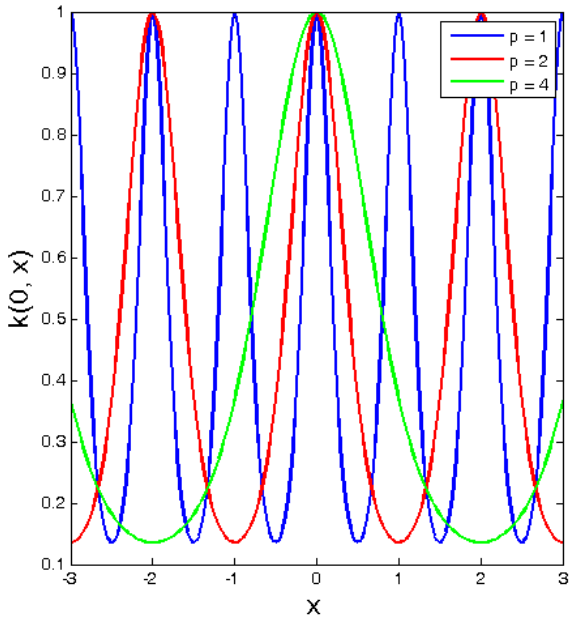
Matlab code: take_rq_samples.m

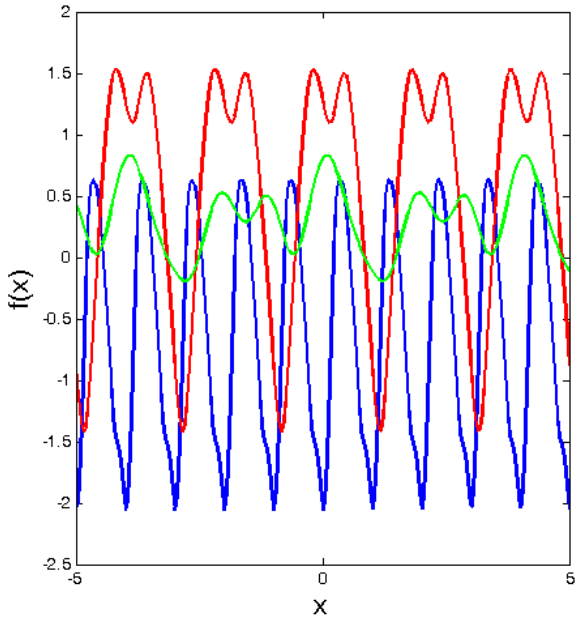
Periodic Covariance

Here is an isotropic covariance function:

$$k_p(x, x') = \exp\left\{-2\sin^2\frac{\left(\frac{\pi|x - x'|}{p}\right)}{\ell^2}\right\},$$

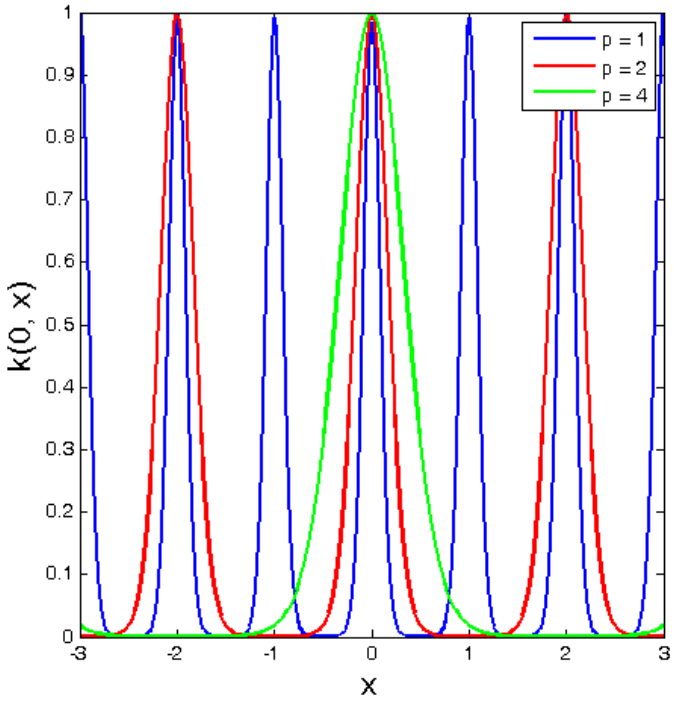
where $s > 0, \ell > 0, p > 0.$

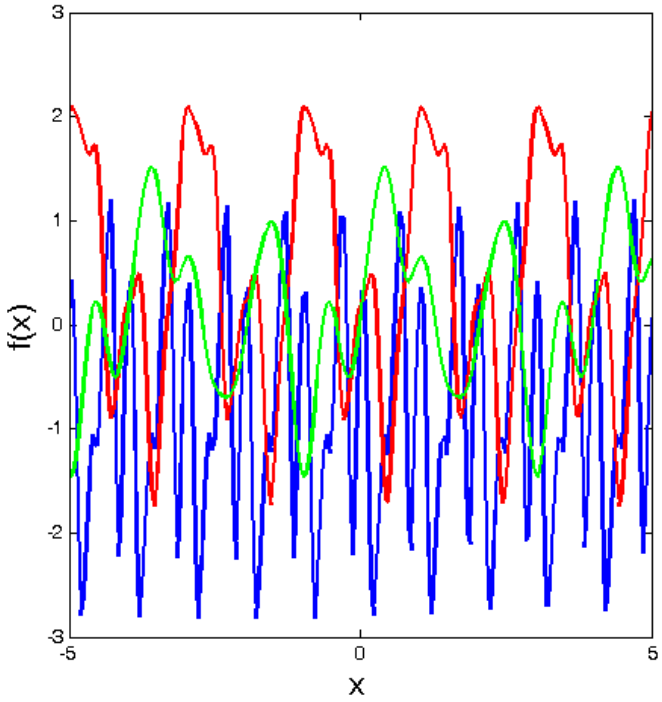




For $\ell = 1.$

Periodic Covariance





For $\ell = 0.5$.

Matlab code: take_periodic_samples.m

Combining Covariance Functions

Here are a few ways you can do this:

- The **sum** of two covariance functions is a covariance function.
- The **product** of two covariance functions is a covariance function.
- The **direct sum** of two covariance functions is a covariance function:
$$k(x, x') = k_1(x_1, x_1') + k_2(x_2, x_2').$$
- The **convolution** of two covariance functions is a covariance function:
$$c(x, x') = \int h(x, z)k(z, z')h(x', z')dzdz'.$$
- The **tensor product** of two covariance functions is a covariance function:
$$k(x, x') = k_1(x_1, x_1')k_2(x_2, x_2').$$

ANOVA Representation

This can help you dealing with high-dimensions if –you are lucky enough that- your function is **additive**:

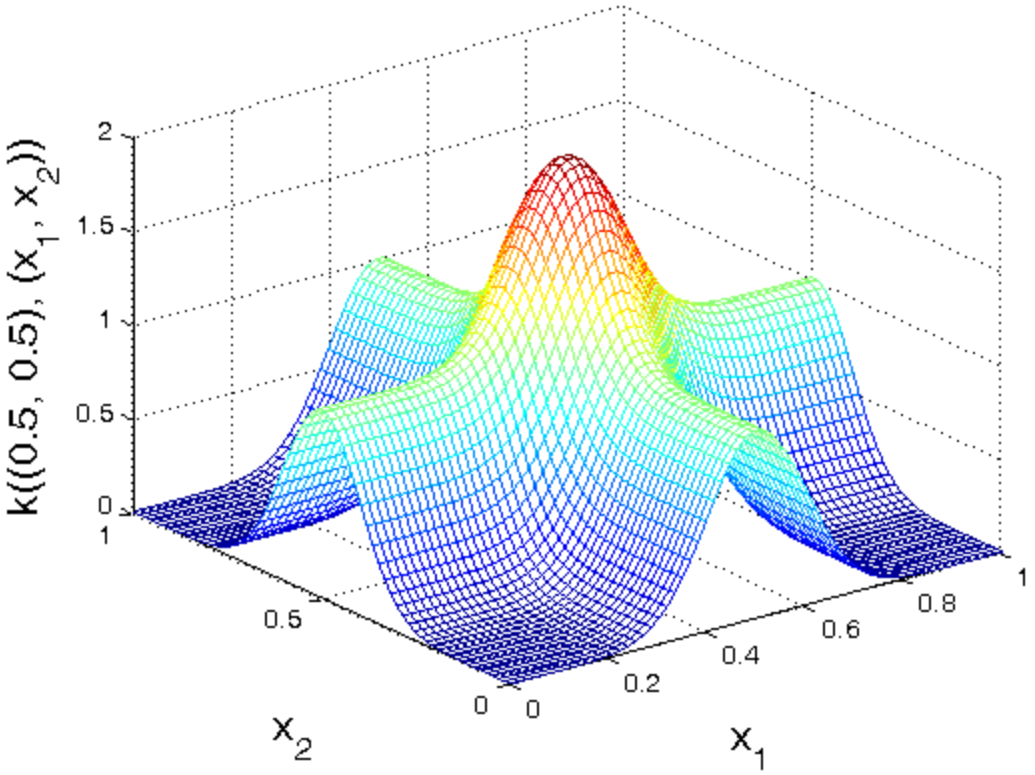
$$f(x) = \sum_{i=1}^{d} f_i(x_i) + \sum_{1 \le i < j \le d} f_{ij}\left(x_i, x_j\right) + \cdots.$$

Such expansions are also known as Analysis Of Variance (ANOVA) or High-Dimensional Model Representation (HDMR).

Theorem: *If the processes f_i, f_{ij}, ... are independent and have covariance functions k_i, k_{ij}, ..., respectively, then the process f has a covariance function of the form:*

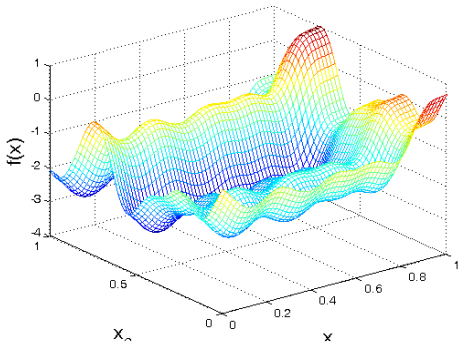
$$k(x, x') = \sum_{i=1}^{d} k_i(x_i, x_i') + \sum_{1 \le i < j \le d} k_{ij}\left(x_i, x_j; x_i', x_j'\right) + \cdots.$$

Example: 1 Interaction

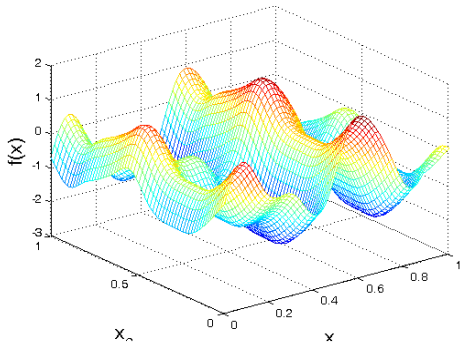


Using a SE with 0.1 length scale and unit signal strength for the dimension wise covariance functions.
Matlab code: anova_demo_1.m

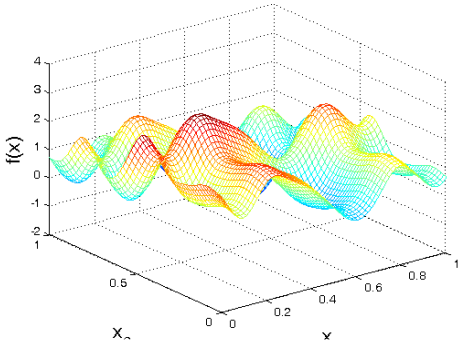
Example: 1 Interaction



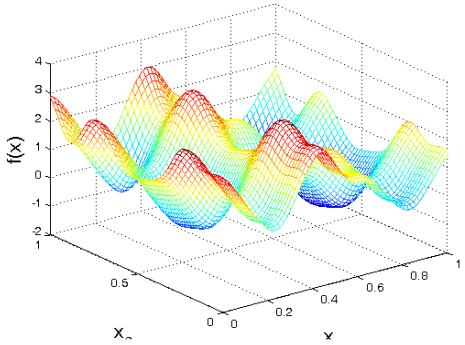
Sample 1



Sample 2

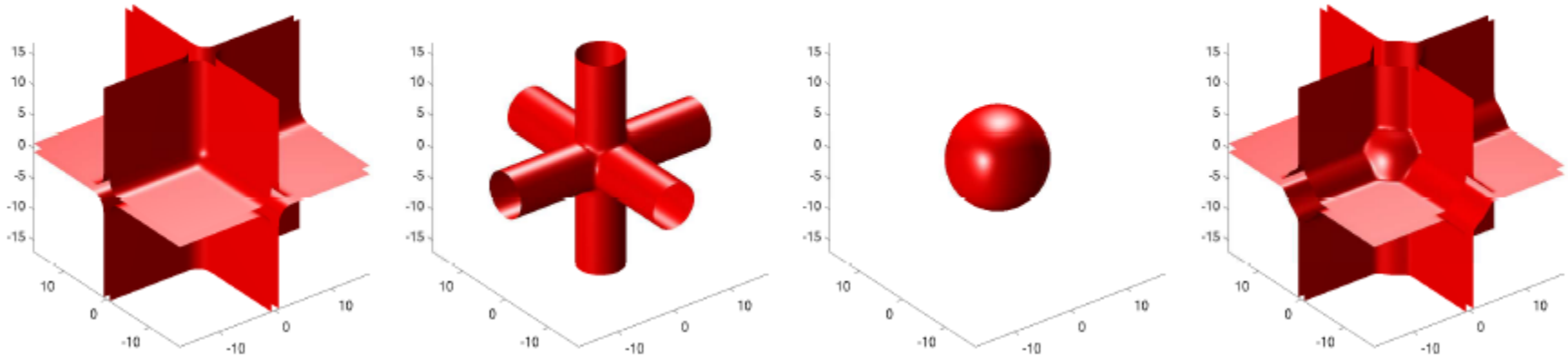


Sample 3



Sample 4

Example: 3D ANOVA



1st order interactions
$k_1 + k_2 + k_3$

2nd order interactions
$k_1 k_2 + k_2 k_3 + k_1 k_3$

3rd order interactions
$k_1 k_2 k_3$
(Squared-exp kernel)

All interactions

(Additive kernel)

Figure 4: Isocontours of additive kernels in 3 dimensions. The third-order kernel only considers nearby points relevant, while the lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

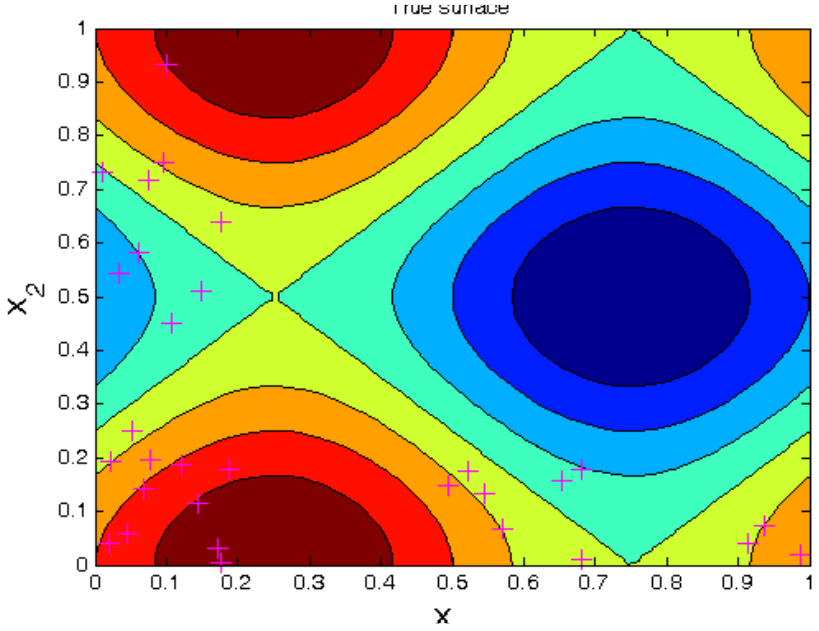
From Duvenau, Nickish, Rasmussen, 2011.

Synthetic Example

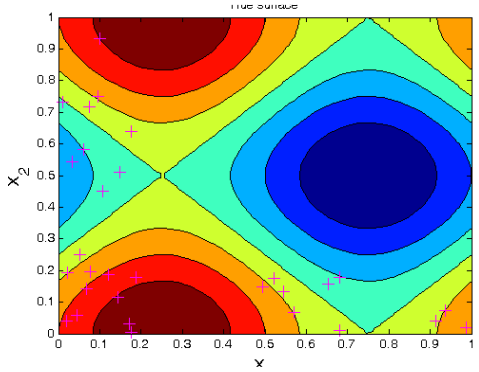
- We look at the function:

$$f(x) = \sin(r\pi x_1) + \cos(r\pi x_2).$$

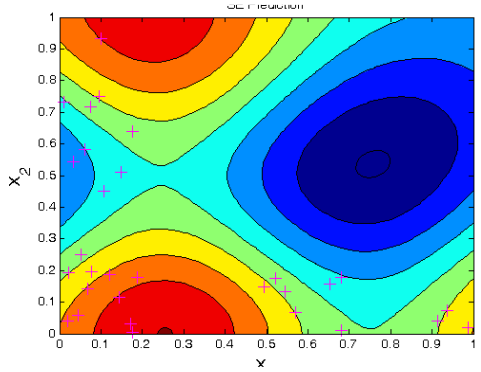
- We observe some random samples of the function in an L-shape region.
- We fit the data using a Square Exponential covariance.
- We fit the data using an ANOVA kernel with 1 term interactions and a Square Exponential additive part.
- Matlab code: anova_demo_2.m.



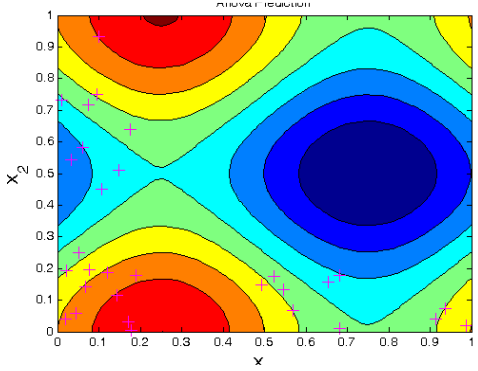
Synthetic Example, r=2



True

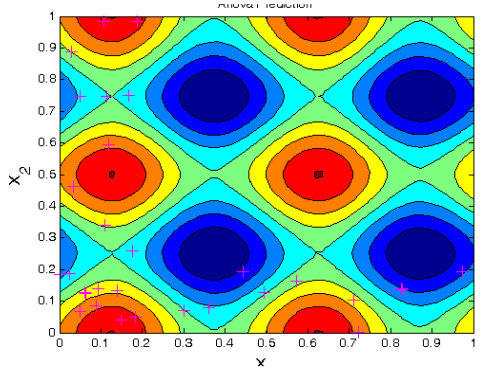


SE

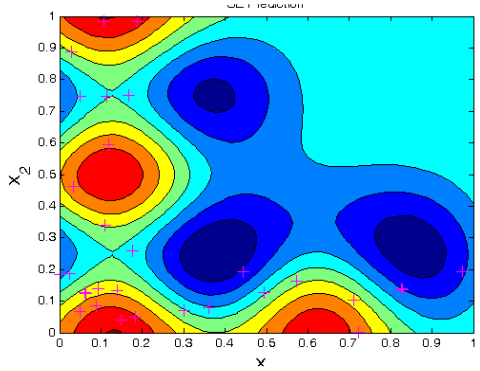


ANOVA-SE 1 term

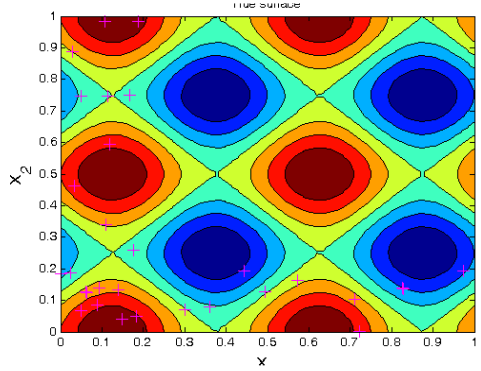
Synthetic Example, r=4



True



SE

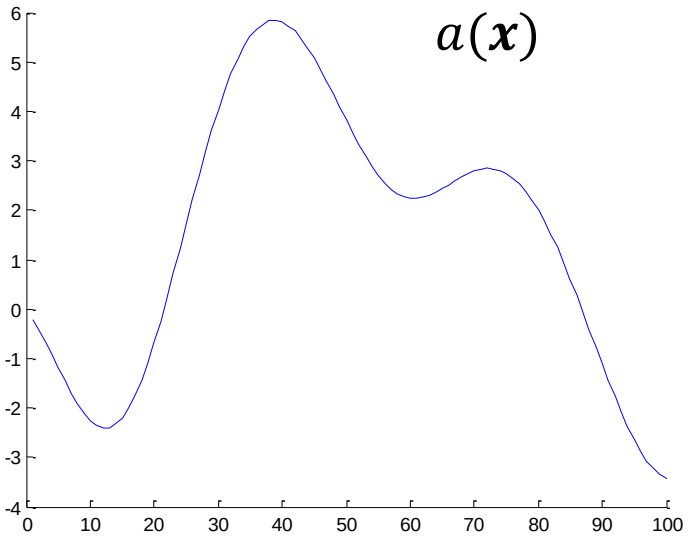


ANOVA-SE 1 term

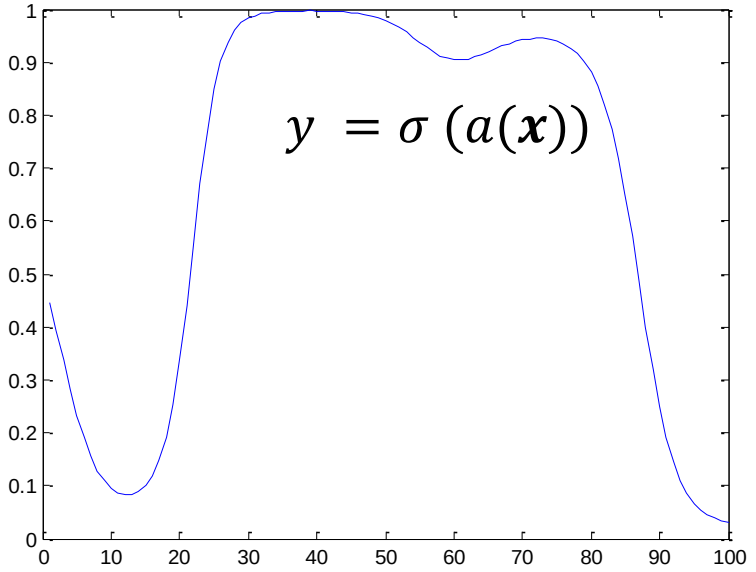
GPs for Classification

Gaussian Processes for Classification

- Objective: model posterior probabilities of the target variable for a new input
- Problem: we need to map values to interval $(0, 1)$
- Solution: use a Gaussian process together with a non-linear activation function (e.g., sigmoid)



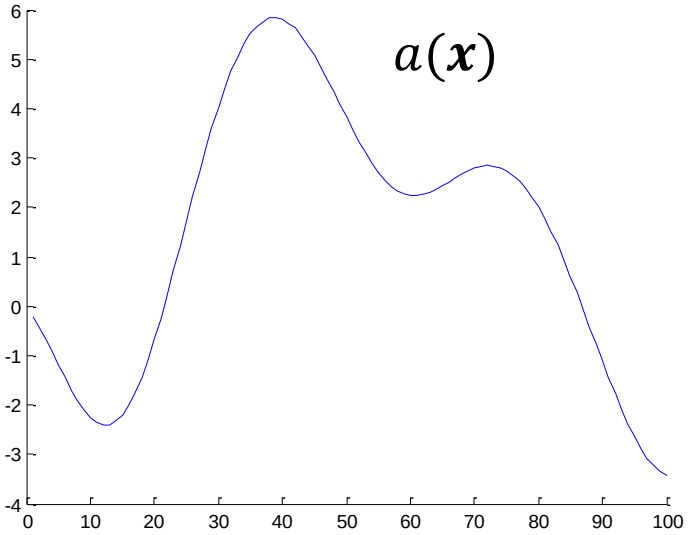
Sample from a Gaussian process over $a(\boldsymbol{x})$

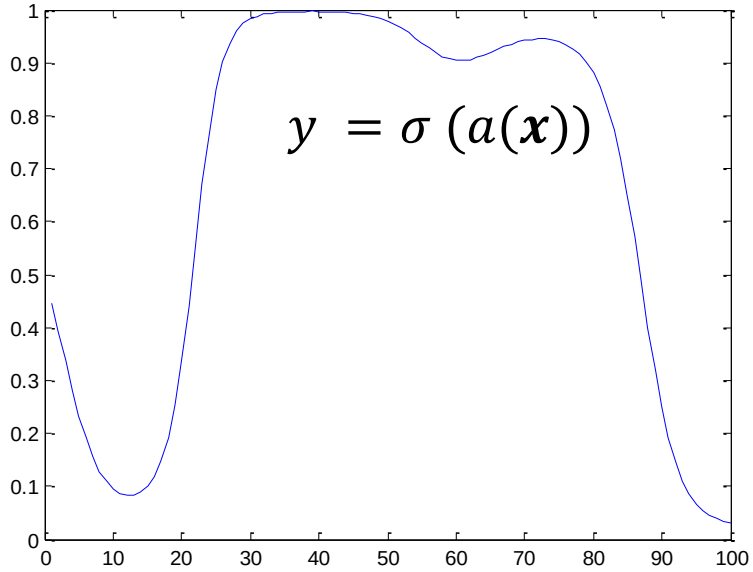


MatLab Code

Gaussian Process for Classification

- Consider a two-class problem with target values $t \in \{0,1\}$. Define a Gaussian process over a function $a(\boldsymbol{x})$ and transform a using the logistic sigmoid to $y = \sigma(a(\boldsymbol{x}))$.
- This will give a non-Gaussian stochastic process over functions $y(\boldsymbol{x})$ where $y \in \{0,1\}$.





MatLab Code

- The probability over the target t is a Bernoulli: $p(t \mid a) = \sigma(a)^t \left(1 - \sigma(a)\right)^{1-t}$

Gaussian Process for Classification

- We denote the training set inputs by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with corresponding observed target variables $\boldsymbol{t} = (t_1, \ldots, t_N)^T$.
- We also consider a single test point $\boldsymbol{x}_{N+1}$ with target value t_{N+1}.
- Our goal is to determine the predictive distribution $p(t_{N+1}|\boldsymbol{t})$, where we left the conditioning on the input variables implicit.
- Introduce a Gaussian process prior over the vector $\boldsymbol{a}_{N+1}$, which has components $a(\boldsymbol{x}_1), \ldots, a(\boldsymbol{x}_{N+1})$.
- This defines a non-Gaussian process over $\boldsymbol{t}_{N+1}$, and by conditioning on $\boldsymbol{t}_N$ we obtain the required predictive distribution $p(t_{N+1}|\boldsymbol{t})$.
- The Gaussian process prior for $\boldsymbol{a}_{N+1}$ takes the form

$$p(\boldsymbol{a}_{N+1}) = \mathcal{N}(\boldsymbol{a}_{N+1} \mid 0, \boldsymbol{C}_{N+1})$$

Gaussian Process for Classification

- The covariance matrix does not include a noise term because we assume that all training data points are correctly labeled.

- However, for numerical reasons it is convenient to introduce a noise-like term governed by a parameter ν that ensures that the covariance matrix is positive definite. Thus the covariance matrix $\boldsymbol{C}_{N+1}$ has elements given by

$$\boldsymbol{C}\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) + \nu\delta_{nm}$$

- The kernel k is positive semidefinite and v is fixed. $k(\boldsymbol{x}, \boldsymbol{x}')$ is governed by parameters θ that we will compute from the training data.

- For two class problems, it is sufficient to predict:

$$p(t_{N+1} = 1 \mid \boldsymbol{t}_N)$$

Gaussian Process for Classification

- The predictive distribution is given as:

$$p(t_{N+1}=1|\boldsymbol{t}_N)=\int p(t_{N+1}=1,a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}$$

$$=\int p(t_{N+1}=1|a_{N+1})\,p(a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}=\int \sigma(a_{N+1})\,p(a_{N+1}|\boldsymbol{t}_N)\,da_{N+1}$$

- The integral is analytically intractable. Can be approximated with sampling (Neal)
- Integration can be analytical if $p(a_{N+1}|t_N)$ can be approximated with a Gaussian (Williams & Barber, Gibbs & MacKay, Gibbs)
- Approximations based on EP can also be used (Opper & Winther, Minka, Seeger)

- Neal, R. M. (1997). Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report 9702, Department of Computer Statistics, University of Toronto.
- Williams, C. K. I. and D. Barber (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 1342–1351.
- Gibbs, M. N. and D. J. C. MacKay (2000). Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks* **11**, 1458–1464.
- Gibbs, M. N. (1997). *Bayesian Gaussian processes for regression and classification*. Phd thesis, University of Cambridge.
- Opper, M. and O. Winther (2000b). Gaussian processes for classification. *Neural Computation* **12**(11), 2655–2684.
- Minka, T. (2001b). *A family of approximate algorithms for Bayesian inference*. Ph. D. thesis, MIT.
- Seeger, M. (2003). *Bayesian Gaussian Process Models: PAC-Bayesian Generalization Error Bounds and Sparse Approximations*. Ph. D. thesis, University of Edinburg.

Gaussian Process for Classification

- We need to compute $p(t_{N+1}=1|\boldsymbol{t}_N)=\int \sigma(a_{N+1})\, p(a_{N+1}|\boldsymbol{t}_N)\, da_{N+1}$
- We have seen in an earlier lecture on logistic regression how to compute the convolution of a Gaussian and a sigmoid function:

$$\int \sigma(a)\mathcal{N}(a|\mu,\sigma^2)da \simeq \sigma\left(\kappa(\sigma^2)\mu\right)$$

$$\kappa(\sigma^2)=\sqrt{1+\frac{\pi\sigma^2}{8}}$$

We can thus try to approximate the posterior distribution $p(a_{N+1}|\boldsymbol{t}_N)$ as Gaussian

- How do we approximate the posterior?
  - Variational inference and make use of the local variational bound on the logistic sigmoid.
  - Expectation propagation
  - Laplace approximation

Laplace Approximation

- We can rewrite the posterior over a_{N+1} using Bayes' theorem:

$$p(a_{N+1} \mid \boldsymbol{t}_N) = \int p(a_{N+1}, \boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N$$

$$= \frac{1}{p(\boldsymbol{t}_N)} \int p(a_{N+1}, \boldsymbol{a}_N)\, p(\boldsymbol{t}_N \mid a_{N+1}, \boldsymbol{a}_N)\, d\boldsymbol{a}_N$$

$$= \frac{1}{p(\boldsymbol{t}_N)} \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N)\, p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, d\boldsymbol{a}_N$$

$$= \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N$$

- We know how to compute mean and covariance for $p(a_{N+1} \mid \boldsymbol{a}_N)$.

$$p(a_{N+1} \mid \boldsymbol{a}_N) = \mathcal{N}\left(\boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N,\, c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}\right)$$

- It remains to find a Gaussian approximation only for $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)$

- This is done noting that $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N)$

Laplace Approximation

- We rewrote the posterior over a_{N+1} using Bayes' theorem:

$$p(a_{N+1} \mid \boldsymbol{t}_N) = \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N$$

where

$$p(a_{N+1} \mid \boldsymbol{a}_N) = \mathcal{N}\left(\boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N, c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}\right)$$

- Also using $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) p(\boldsymbol{a}_N)$, $p(\boldsymbol{a}_N) = \mathcal{N}\left(0, \boldsymbol{C}_N\right)$ and

$$p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) = \prod_{n=1}^{N} \sigma(a_n)^{t_n} \left(1 - \sigma(a_n)\right)^{1-t_n} = \prod_{n=1}^{N} e^{a_n t_n} \sigma(-a_n)$$

where in the last derivation we used: $\sigma(a_n) = \dfrac{1}{1+e^{-a_n}}$ and

$$\sigma(a_n)^{t_n} \left(1 - \sigma(a_n)\right)^{1-t_n} = \underbrace{\left(1 - \sigma(a_n)\right)}_{\sigma(-a_n)} \left(\underbrace{\frac{\sigma(a_n)}{1 - \sigma(a_n)}}_{e^{a_n}} \right)^{t_n} = e^{a_n t_n} \sigma(-a_n)$$

Laplace Approximation

- The log of $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) p(\boldsymbol{a}_N)$ which up to an additive constant is:

$$p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \propto p(\boldsymbol{a}_N) p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)$$

$$\propto \mathcal{N}(0, \boldsymbol{C}_N) \prod_{n=1}^{N} e^{a_n t_n} \underbrace{\sigma(-a_n)}_{1/(1+e^{a_n})} \Rightarrow$$

$$\Psi(\boldsymbol{a}_N) = \ln p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) = \ln p(\boldsymbol{t}_N \mid \boldsymbol{a}_N) + \ln p(\boldsymbol{a}_N)$$

$$= -\frac{1}{2}\boldsymbol{a}_N^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T \boldsymbol{a}_N - \sum_{n=1}^{N} \ln\left(1 + e^{a_n}\right)$$

- We will need to compute the 2nd derivative of this to come up with the Laplace approximation.

Laplace Approximation

$$\Psi(\boldsymbol{a}_N) = -\frac{1}{2}\boldsymbol{a}_N^T\boldsymbol{C}_N^{-1}\boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N - \sum_{n=1}^{N}\ln\left(1+e^{a_n}\right)$$

- The gradients are given as

$$\nabla\Psi(\boldsymbol{a}_N) = \boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N, \quad \boldsymbol{\sigma}_N = \left(\sigma(a_1),...,\sigma(a_N)\right)^T$$

$$\nabla^2\Psi(\boldsymbol{a}_N) = -\boldsymbol{W}_N - \boldsymbol{C}_N^{-1}, \quad \boldsymbol{W}_N = diag\left(\sigma(a_1)(1-\sigma(a_1)),...,\sigma(a_N)(1-\sigma(a_N))\right)$$

- We cannot find the mode by setting $\nabla\Psi(\boldsymbol{a}_N) = 0$ since $\boldsymbol{\sigma}_N$ depends nonlinearly on $\boldsymbol{a}_N$. For this we will use the IRLS algorithm.
- Note that $0 < \sigma(a_1)(1-\sigma(a_1)) < 1/4 \Rightarrow \boldsymbol{W}_N$ is positive definite.
- The Hessian $A = -\nabla^2\Psi(\boldsymbol{a}_N)$ is positive definite and thus the posterior $p(\boldsymbol{a}_N|\boldsymbol{t}_N)$ is log convex and has a single global mode.
- At the mode, the gradient vanishes giving: $\boldsymbol{a}_N^* = \boldsymbol{C}_N(\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$

Laplace Approximation

- We can use the iterative reweighted least squares (IRLS) algorithm ($\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$) to update for $\boldsymbol{a}_N$ (find the minimum of $-\ln p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)$).

- The iterative update equation for $\boldsymbol{a}_N$ is:

$$\boldsymbol{a}_N^{new} = \boldsymbol{C}_N(\boldsymbol{I} + \boldsymbol{W}_N\boldsymbol{C}_N)^{-1}\left(\boldsymbol{t}_n - \boldsymbol{\sigma}_n + \boldsymbol{W}_N\boldsymbol{a}_N\right)$$

Here we used

$$\boldsymbol{a}_N^{new} = \boldsymbol{a}_N + (\boldsymbol{W}_N + \boldsymbol{C}_N^{-1})^{-1}\left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right) = \boldsymbol{a}_N + \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right)$$

$$= \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left\{(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})\boldsymbol{C}_N^{-1}\boldsymbol{a}_N + \left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N - \boldsymbol{C}_N^{-1}\boldsymbol{a}_N\right)\right\}$$

$$= \boldsymbol{C}_N(\boldsymbol{W}_N\boldsymbol{C}_N + \boldsymbol{I})^{-1}\left\{\boldsymbol{W}_N\boldsymbol{a}_N + \boldsymbol{t}_N - \boldsymbol{\sigma}_N\right\}$$

Laplace Approximation

- We can use the iterative reweighted least squares (IRLS) algorithm ($\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$) to update for $\boldsymbol{a}_N$.

- The iterative update equation for $\boldsymbol{a}_N$ is:

$$\boldsymbol{a}_N^{new} = \boldsymbol{C}_N(\boldsymbol{I} + \boldsymbol{W}_N\boldsymbol{C}_N)^{-1}\left(\boldsymbol{t}_n - \boldsymbol{\sigma}_n + \boldsymbol{W}_N\boldsymbol{a}_N\right)$$

- The mode position $\boldsymbol{a}_N^*$ and the Hessian matrix $\boldsymbol{H}$ at this position $\boldsymbol{H} = -\nabla^2\Psi(\boldsymbol{a}_N) = \boldsymbol{W}_N + \boldsymbol{C}_N^{-1}$ define the Gaussian approximation to $p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)$

$$q(\boldsymbol{a}_N) = \mathcal{N}(\boldsymbol{a}_N \mid \boldsymbol{a}_N^*, \boldsymbol{H}^{-1})$$

In the Hessian note that $\boldsymbol{W}_N$ is computed using $\boldsymbol{a}_N^*$.

Laplace Approximation

- Now we can go back to the formulas for linear Gaussian models and compute the integrals

$$p(a_{N+1} \mid \boldsymbol{t}_N) = \int p(a_{N+1} \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{t}_N)\, d\boldsymbol{a}_N$$

$$p(a_{N+1} \mid \boldsymbol{a}_N) = \mathcal{N}\left(\boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N, c - \boldsymbol{k}^T \boldsymbol{C}_N^{-1} \boldsymbol{k}\right)$$

$$p(\boldsymbol{a}_N \mid \boldsymbol{t}_N) \approx q(\boldsymbol{a}_N) = \mathcal{N}(\boldsymbol{a}_N \mid \boldsymbol{a}_N^*, \boldsymbol{H}^{-1}),\ \boldsymbol{H} = \boldsymbol{W}_N + \boldsymbol{C}_N^{-1}$$

from which we have:

$$p(a_{N+1} \mid \boldsymbol{t}_N) = \mathcal{N}\left(\boldsymbol{k}^T (\boldsymbol{t}_N - \boldsymbol{\sigma}_N), c - \boldsymbol{k}^T \left(\boldsymbol{W}_N^{-1} + \boldsymbol{C}_N\right)^{-1} \boldsymbol{k}\right)$$

- Finally, we can also compute $p(t_{N+1} \mid \boldsymbol{t})$ (or the decision boundary $p(t_{N+1} \mid \boldsymbol{t}) = 0.5$ – in this case $\mu = \boldsymbol{k}^T (\boldsymbol{t}_N - \boldsymbol{\sigma}_N) = 0$)

$$p(t_{N+1} = 1 \mid \boldsymbol{t}_N) = \int \sigma(a_{N+1})\, p(a_{N+1} \mid \boldsymbol{t}_N)\, da_{N+1}$$

using $$\int \sigma(a)\mathcal{N}(a \mid \mu, \sigma^2)\, da \approx \sigma\left(\kappa(\sigma^2)\mu\right), \quad \kappa(\sigma^2) = \sqrt{1 + \frac{\pi\sigma^2}{8}}$$

Learning the Hyperparameters

- To determine the parameters $\boldsymbol{\theta}$ of the covariance function, we can maximize the likelihood function:

$$p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \int p(\boldsymbol{t}_N \mid \boldsymbol{a}_N)\, p(\boldsymbol{a}_N \mid \boldsymbol{\theta}) d\boldsymbol{a}_N = \int p(\boldsymbol{t}_N, \boldsymbol{a}_N \mid \boldsymbol{\theta}) d\boldsymbol{a}_N$$

- The integral is analytically intractable. The Laplace approximation can be applied again:

$$\ln p(\boldsymbol{t}_N, \boldsymbol{a}_N) = \Psi(\boldsymbol{a}_N) = -\frac{1}{2}\boldsymbol{a}_N^T \boldsymbol{C}_N^{-1} \boldsymbol{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T \boldsymbol{a}_N - \sum_{n=1}^{N} \ln\left(1 + e^{a_n}\right)$$

$$\ln p(\boldsymbol{t}_N, \boldsymbol{a}_N) \approx \Psi(\boldsymbol{a}_N^*) - \frac{1}{2}(\boldsymbol{a}_N - \boldsymbol{a}_N^*)^T (W_N + C_N^{-1})(\boldsymbol{a}_N - \boldsymbol{a}_N^*)$$

- Integrating in $\boldsymbol{a}_N$ and using the normalization factor of a Gaussian gives:

$$\ln p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \boldsymbol{\Psi}(\boldsymbol{a}_N^*) - \frac{1}{2}\ln|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}| + \frac{N}{2}\ln(2\pi)$$

where

$$\boldsymbol{\Psi}(\boldsymbol{a}_N^*) = \ln p(\boldsymbol{a}_N^* \mid \boldsymbol{\theta}) + \ln p(\boldsymbol{t}_N \mid \boldsymbol{a}_N^*)$$

- We now need an expression for the gradient of the log of $p(\boldsymbol{t}_N|\boldsymbol{\theta})$. Both $\boldsymbol{a}_N^*$ and $\boldsymbol{C}_N$ depend on $\boldsymbol{\theta}$.

Learning the Hyperparameters

- To determine the parameters , maximize the likelihood:

$$\Psi(\boldsymbol{a}_N^*) = -\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N^* - \sum_{n=1}^{N}\ln\left(1+e^{a_n^*}\right)$$

$$\ln p(\boldsymbol{t}_N|\boldsymbol{\theta}) = \Psi\left(\boldsymbol{a}_N^*\right) - \frac{1}{2}\ln|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}| + \frac{N}{2}\ln(2\pi)$$

- Using $\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\right)$ and $\frac{\partial \boldsymbol{A}^{-1}}{\partial x} = -\boldsymbol{A}^{-1}\frac{\partial \boldsymbol{A}}{\partial x}\boldsymbol{A}^{-1}$, we obtain

the following terms of $\frac{\partial}{\partial \theta_j}\ln p(\boldsymbol{t}_N|\theta)$ with explicit dependence on θ:

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\boldsymbol{C}_N^{-1}\left[\boldsymbol{I} - \left(\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right)^{-1}\boldsymbol{C}_N^{-1}\right]\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right] =$$

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\boldsymbol{C}_N^{-1}\left[\left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)\left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)^{-1} - \left(\boldsymbol{C}_N\boldsymbol{W}_N + \boldsymbol{I}\right)^{-1}\right]\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right]$$

$$\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}Tr\left[\left(I + \boldsymbol{C}_N\boldsymbol{W}_N\right)^{-1}\boldsymbol{W}_N\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right]$$

Learning the Hyperparameters

- To determine the parameters, maximize the likelihood:

$$\Psi(\boldsymbol{a}_N^*) = -\frac{1}{2}\boldsymbol{a}_N^{*T}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \boldsymbol{t}_N^T\boldsymbol{a}_N^* - \sum_{n=1}^{N}\ln\left(1+e^{a_n^*}\right)$$

$$\ln p(\boldsymbol{t}_N \mid \boldsymbol{\theta}) = \Psi\left(\boldsymbol{a}_N^*\right) - \frac{1}{2}\ln\left|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right| + \frac{N}{2}\ln(2\pi)$$

- Using $\nabla\Psi(\boldsymbol{a}_N^*) = 0$, the term in $\frac{\partial}{\partial\theta_j}\ln p(\boldsymbol{t}_N \mid \theta)$ with explicit dependence on $\boldsymbol{a}_n^*$ is as follows:

$$-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial\ln\left|\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right|}{\partial a_n^*}\frac{\partial a_n^*}{\partial\theta_j}$$

- Using $\frac{\partial}{\partial x}\ln|\boldsymbol{A}| = Tr\left(\boldsymbol{A}^{-1}\frac{\partial\boldsymbol{A}}{\partial x}\right)$ and $\boldsymbol{W}_N = diag\left(\sigma(a_i)(1-\sigma(a_i))\right)$, $d\sigma/da_i = \sigma(a_i)(1-\sigma(a_i))$, it becomes

$$-\frac{1}{2}\sum_{n=1}^{N}Tr\left[\left(\boldsymbol{W}_N + \boldsymbol{C}_N^{-1}\right)^{-1}\underbrace{\frac{\partial\boldsymbol{W}_N}{\partial\sigma(a_n)}}_{(1-2\sigma_n^*)}\underbrace{\frac{\partial\sigma(a_n)}{\partial a_n}}_{\sigma_n^*(1-\sigma_n^*)}\right]\frac{\partial a_n^*}{\partial\theta_j} = -\frac{1}{2}\sum_{n=1}^{N}\left[\left(I + \boldsymbol{C}_N\boldsymbol{W}_N\right)^{-1}\boldsymbol{C}_N\right]_{nn}\underset{\sigma(a_n^*)}{\sigma_n^*}(1-\sigma_n^*)(1-2\sigma_n^*)\frac{\partial a_n^*}{\partial\theta_j}$$

Learning Hyperparameters

- We finally can compute $\partial a_n^* / \partial \theta_j$ as follows : $\boldsymbol{a}_N^* = \boldsymbol{C}_N \left(\boldsymbol{t}_N - \boldsymbol{\sigma}_N \right) \Rightarrow$

$$\frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j} = \frac{\partial \boldsymbol{C}_N}{\partial \theta_j} (\boldsymbol{t}_N - \boldsymbol{\sigma}_N) - \boldsymbol{C}_N \boldsymbol{W}_N \frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j}$$

$$\boldsymbol{W}_N = diag\left(\sigma(a_i)\left(1-\sigma(a_i)\right) \right)$$

$$d\sigma / da_i = \sigma(a_i)(1-\sigma(a_i))$$

- From this we derive:

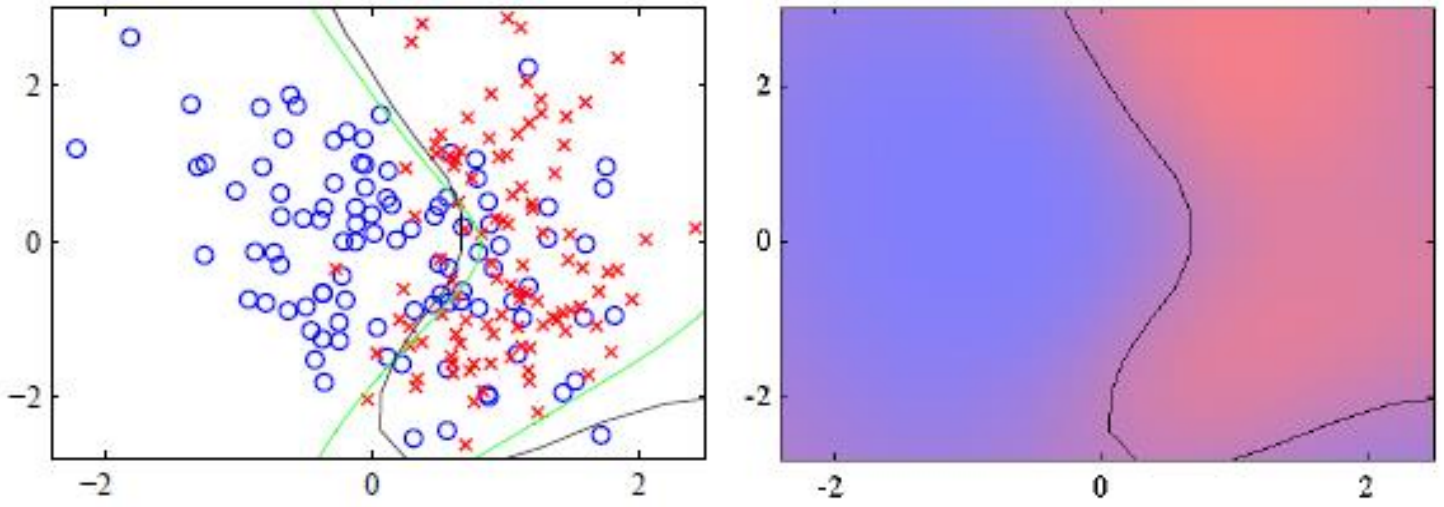
$$\frac{\partial \boldsymbol{a}_N^*}{\partial \theta_j} = \left(\boldsymbol{I} + \boldsymbol{C}_N \boldsymbol{W}_N \right)^{-1} \frac{\partial \boldsymbol{C}_N}{\partial \theta_j} (\boldsymbol{t}_N - \boldsymbol{\sigma}_N)$$

- We now have all the terms in the gradient $\frac{\partial}{\partial \theta_j} \ln p(\boldsymbol{t}_N \mid \theta)$ and can employ gradient optimization techniques.

Binary Classifier Based on Gaussian Process

- We now have a binary classifier based on a Gaussian process. On left the data together with the optimal decision boundary from the true distribution in green, and the decision boundary from the Gaussian process classifier in black.

- On right the predicted posterior probability for the blue and red classes together with the Gaussian process decision boundary.



Requires installing NetLab and setting the appropriate path

MatLab Code

- Williams, C. K. I. and D. Barber (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**, 1342–1351.

Connections to Neural Networks

- Neural Networks
  - The range of representable functions is governed by the number M of hidden units
  - Within the maximum likelihood framework, they overfit as M comes close to the number of training samples
- Bayesian Neural Networks
  - The prior over $\boldsymbol{w}$ in conjunction with the network function $f(\boldsymbol{x}, \boldsymbol{w})$ produces a prior distribution over functions from $y(\boldsymbol{x})$
  - The distribution of functions will tend to a GP as $M \rightarrow \infty$.
    - In this limit the output variables of the neural network become independent.
    - However, the property that the outputs share hidden units and thus all of them affecting the values of the weights is lost in this limit.

- Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer. Lecture Notes in Statistics 118.

Connections to Neural Networks

- There are explicit forms for the covariance in the case of probit and Gaussian hidden unit activation functions.
  - These kernel functions $k(\boldsymbol{x}, \boldsymbol{x}')$ are nonstationary, i.e. not a function of $\boldsymbol{x} - \boldsymbol{x}'$.
  - This is a consequence of the Gaussian weight prior being centered on zero which breaks translation invariance in weight space.
- By working directly with the covariance function we have implicitly marginalized over the distribution of weights.
- If the weight prior is governed by hyperparameters, then their values will determine the length scales of the distribution over functions.

- Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Computation* **10**(5), 1203–1216.